Zotonic

Release 0.32.0

Sep 04, 2017

Contents

Welcome to the Zotonic documentation! This describes Zotonic 0.32, last updated Sep 04, 2017.

The latest docs are also available in a PDF version.

CHAPTER 1

User Guide

An non-technical introduction to the Zotonic CMS for end users.

User Guide

Welcome to the Zotonic User Guide. This guide is a non-technical introduction to Zotonic.

Table of contents

CMS

Zotonic is a Content Management System (CMS).

Todo

extend CMS introduction

The Zotonic data model

Zotonic's data model can be seen as a pragmatic implementation of the Semantic Web: a mixture between a traditional database and a triple store.

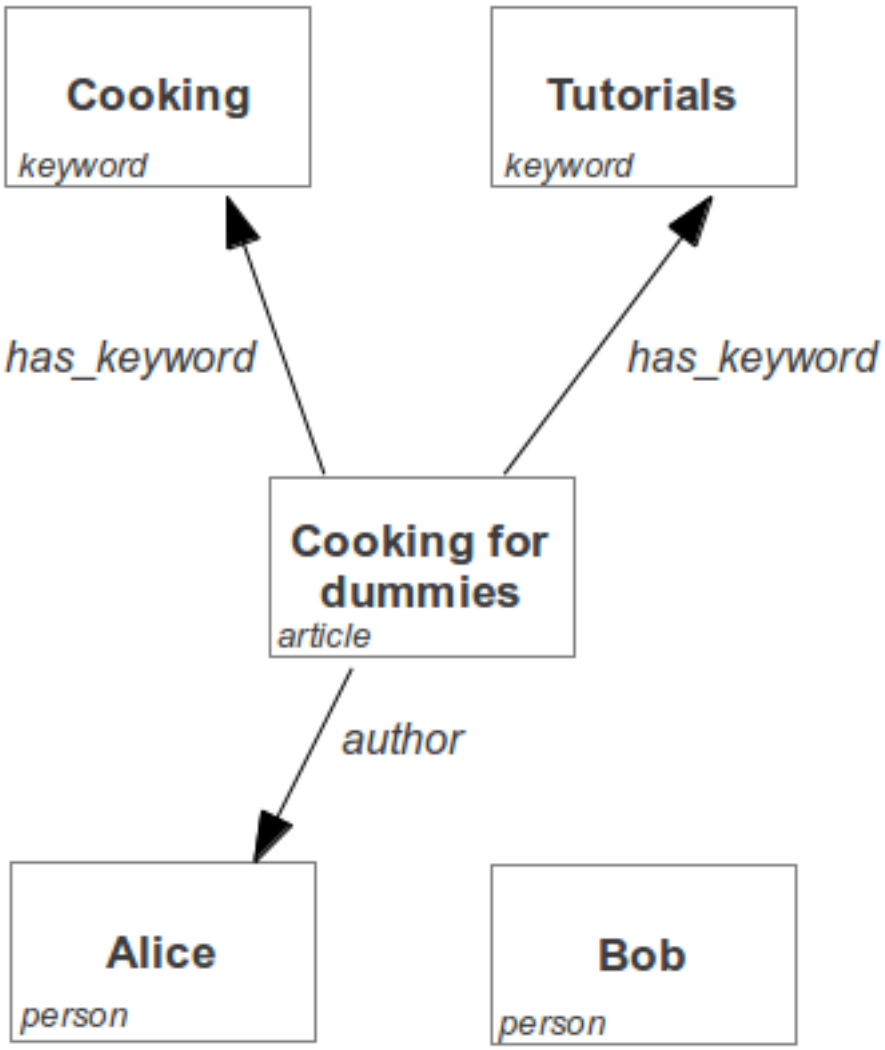
The data model has two main concepts: the *resource* and the *edge*.

Resources, which are often called *pages* in the admin, are the main data unit: they have properties like title, summary, body text; and importantly, they belong to a certain *category*.

Edges are nothing more than connections between two resources. Each edge is labeled, with a so-called *predicate*.

This manual describes the data model and its related database table in depth.

The data model by example



Take the example data on the right. It shows resources, connected to each other by edges. The rectangular blocks denote the resources, the arrow between them denote the edges.

It shows the *domain model* of a basic blog-like structure: *article* resources which are written by *persons* and articles being tagged with keywords.

The edge between *Alice* and *cooking for dummies* is labelled *author*: indicating that Alice wrote that article.

Note that these edges have a direction. When reasoning about edges, it is the easiest to think of them in terms of grammar: Subject - verb - object. In the case of Alice:

- **Cooking for dummies** is authored by **Alice**;
- **Cooking for dummies** has the keyword **Cooking**;

or more general:

- **subject predicate object**.

In Zotonic, the terms *subject* and *object* (shortened as *s* and *o*) are used in the templates, allowing you to traverse the edges between resources:

```
{{ m.rsc[id].o.author[1].title }}
```

Returns the name of the first author of the article with the given *id*, by traversing to the first *author* object edge of the given id. See the *m_rsc* (page 426) model for more details on this.

Pages

Pages are the main building block of Zotonic's *data model* (page 3). For simplicity of communication, a resource is often referred to as a page. Every resource usually has its own page on the web site.

By default, each page has the following properties.

id	Unique number that identifies the resource and can be used for referring to the resource
title	Main title of the page
summary	Short, plain-text, summary
short title	Short version of the title that is used in navigation and other places where an abbreviated title is shown
published	Only published pages are visible to the general public
publication start	Date at which the page will be published and
publication end	Data at which the page will be unpublished
name	Alternative for id: used to uniquely identify the resource
page path	The page's URL on the site. Defaults to `page/{id}/{title}`

See also:

resources (page 16) in the Developer Guide

Categories

Each page belongs to one category. The category a page is in determines how it is displayed.

Edges

An *edge* is a labeled connection between two resources.

The `edge` table defines these relations between resources. It does this by adding a directed edge from one rsc record (subject) to another (object). It also adds a reference to the *predicate*: the label of the edge.

In the admin, edges are represented in the "Page connections" sidebar panel, of the edit page of the *subject*: the resource where the edges originate. By convention, edges are said to *belong* to their subject. This is to simplify the access control: if you are allowed to edit the resource, you're also allowed to edit its *outgoing* edges ("Page connections" in the admin), creating connections to other resources.

See also:

m_edge (page 417)

Predicates

Edges have a label: like in *The data model by example* (page 4), *author* is a *predicate* of an edge which denotes that a certain *article* was written by a certain *person*

Just like categories, these predicates are themselves also resources: allowing you to specify metadata, give them a meaningful title, et cetera.

Each predicate has a list of valid subject categories and valid object categories (stored in the `predicate_category` table). This is used to filter the list of predicates in the admin edit page, and also to filter the list of found potential objects when making a connection. On their edit page in the admin interface, you can edit the list of valid subject and object categories for a predicate.

See also:

m_predicate (page 424)

Further reading

- Zotonic's defaults resources and categories: the *domain model* (page 6).

Domain model

By default, Zotonic comes with a set of categories and predicates that makes sense for typical websites.

You can extend the built-in domain model by adding your own categories and predicates.

Todo

document categories, predicates and resources

User management

Create new users

In the admin, under 'Auth' > 'Users' you can find a list of all users. Use the 'Make a new user' button to create a new user with a username/password identity.

Note: This requires *mod_admin_identity* (page 281) to be enabled.

Issues and features

If you encounter any issues in using Zotonic, or have ideas for new features, please let us know at the Zotonic issue tracker. You need to have a free GitHub account to do so.

Before submitting a new issue, please search the issues list to see if the issue bug has already been reported.

Every first Monday of the month a new version of Zotonic is *released* (page 76).

Todo

Add more chapters to have a complete run-through of all user-facing aspects of Zotonic.

See http://zotonic.com/support for getting help and support.

See also:

if you're a developer who's already familiar with Zotonic, you may want to read the *Developer Guide* (page 7) instead.

CHAPTER 2

Developer Guide

The handbook for all Zotonic developers. It guides you through all aspects of the framework.

Developer Guide

Introduction

This is the Zotonic Developer Guide. It takes you through all aspects of Zotonic so you can start building your own sites as quickly as possible.

If you are an end user rather than developer, you may want to read the User Guide instead.

An overview of Zotonic

Zotonic is a complete stack for building advanced websites quickly. It consists of:

1. a content management system (CMS)
2. a web framework
3. a web server.

You can rapidly develop CMS-based sites on top of Zotonic. If you need more flexibility, you can customise all aspects of Zotonic. The included web server means can start working with Zotonic right away, without the need for installing Apache or Nginx separately.

Zotonic is structured into modules. Each module offers a specific piece of functionality; the CMS is itself a module. There are modules for access control, social media integration, TLS, video embeds and much more.

Central to Zotonic is its uniquely flexible data model.

Installation

To get Zotonic running, you first need to install it. This chapter describes how to install Zotonic manually. Alternatively, you can run Zotonic from one of the *Docker images* (page 9).

Preparation

First prepare your system for running Zotonic. Zotonic needs:

- Erlang 18 or higher
- PostgreSQL 8.4 or higher
- ImageMagick 6.5 or higher for image resizing
- Git for pulling in external dependencies

See also:

a more extensive discussion of *all requirements* (page 488)

On Linux

For instance on Debian you can install the dependencies by running:

```
$ sudo apt-get install build-essential git erlang imagemagick postgresql
```

On OS X

Install Homebrew, then run:

```
$ brew install erlang git imagemagick postgresql
```

On Windows

Currently, Zotonic is not officially supported on the Windows platform. However, the main dependencies Erlang, PostgreSQL and ImageMagick do work on Windows, so, if you're adventurous, it should be possible to get it running.

We have included user-contributed `start.cmd` and `build.cmd` batch-scripts which used to work on Windows, but have not been kept up-to-date with recent changes. Expect some major tweaking to get this back on track.

Getting Zotonic

1. Download the latest Zotonic release ZIP file from the GitHub releases page. For instance:

   ```
   $ wget https://github.com/zotonic/zotonic/archive/0.32.0.zip
   ```

   Then unzip the file and rename the directory:

   ```
   $ unzip 0.32.0.zip
   $ mv zotonic-0.32.0 zotonic
   ```

   Alternatively, clone the latest development version using Git:

   ```
   $ git clone https://github.com/zotonic/zotonic.git
   ```

2. You then need to compile the Zotonic sources:

   ```
   $ cd zotonic
   $ make
   ```

3. Then start Zotonic in debug mode:

```
$ bin/zotonic debug
```

4. Now point your browser to: http://localhost:8000/. You should see a welcome message, 'Powered by Zotonic'. This is the so-called *status website* (page 494). So far, so good! Now it's time to *create your first site* (page 10).

Next steps

- *Create your first site* (page 10).
- Log in to the *status site* (page 494).
- If something goes wrong, read the *troubleshooting reference* (page 498).
- Read more about Zotonic *configuration* (page 490).

Docker

We offer three Docker images:

- zotonic/zotonic-full contains both Zotonic and PostgreSQL. Use this to get started quickly if you want to build sites on Zotonic.
- zotonic/zotonic contains only Zotonic. This image is most useful in production setups or when you're using Docker Compose with a separate database container.
- zotonic/zotonic-dev contains build tools and Erlang. Use this image for development work on Zotonic itself, where you'll mount your Zotonic directory as a volume in the container.

To use any of the images, first download and install Docker.

Start a Zotonic image on your Docker machine:

```
# use a stable version:
$ docker run -d -p 8000:8000 zotonic/zotonic-full:0.17.0

# or a branch:
$ docker run -d -p 8000:8000 zotonic/zotonic-full:0.x

# or run the latest version from master:
$ docker run -d -p 8000:8000 zotonic/zotonic-full:latest
```

Mount a volume that contains your Zotonic sites:

```
$ docker run -d -v `pwd`/sites:/opt/zotonic/user/sites zotonic/zotonic-full
```

And mount a volume with your custom Zotonic modules:

```
$ docker run -d -v `pwd`/sites:/opt/zotonic/user/sites -v `pwd`/modules:/opt/
↪zotonic/user/modules zotonic/zotonic-full
```

zotonic-dev

You can use the zotonic/zotonic-dev image when you're doing development work on Zotonic. Start the container from your local Zotonic clone:

```
$ git clone https://github.com/zotonic/zotonic.git
```

To start the container, use Docker Compose:

```
$ docker-compose run --service-ports zotonic bash
```

This opens a shell prompt in the Zotonic container. A PostgreSQL container will be started automatically as well. In the Zotonic container, you can enter any command. So, to start Zotonic in debug mode:

```
$ bin/zotonic debug
```

The `--service-ports` flags exposes Zotonic's port 8000 as your local port 80, so you can view the *Zotonic status page* (page 494) at `http://localhost`.

You can also run other commands in the container, such as running the tests:

```
$ bin/zotonic runtests
```

Any changes you make in the Zotonic source files will be propagated to the container and *automatically compiled* (page 290).

Sites

Zotonic has the capability of serving more than one site at a time. You can have multiple sites enabled, each with its own set of templates, database and dispatch rules. Each site has its own hostname.

Create a site

1. First, prepare the database. In your terminal, connect to PostgreSQL:

   ```
   $ sudo -u postgres psql (enter your OS password)
   ```

   And create a database user for Zotonic. You may want to change the password:

   ```
   postgres=# CREATE USER zotonic WITH PASSWORD 'zotonic';
   ```

   Now, either give this user create rights to have Zotonic automatically create the database for you (recommended):

   ```
   postgres=# ALTER USER zotonic CREATEDB;
   ```

   Or create the site database manually:

   ```
   postgres=# CREATE DATABASE zotonic WITH OWNER = zotonic ENCODING = 'UTF8';
   postgres=# GRANT ALL ON DATABASE zotonic TO zotonic;
   postgres=# \c zotonic
   postgres=# CREATE LANGUAGE "plpgsql";
   ```

2. Edit your `/etc/hosts` file, adding an entry for `yoursite.dev` (the site hostname) to point at your local host:

   ```
   127.0.0.1   yoursite.dev
   ```

3. Finally, create a new Zotonic site, based on the 'blog' skeleton site:

   ```
   $ bin/zotonic addsite -s blog yoursite
   ```

   Note: Zotonic has to be running for the `addsite` command to succeed.

4. Then rebuild Zotonic:

```
$ cd dir/to/zotonic
$ make
```

5. And (re)start Zotonic:

```
$ bin/zotonic debug
```

5. Finally, point your browser to http://yoursite:8000 to see your new site. You can log into the admin at http://yoursite:8000/admin with the password that you can find in your site's configuration file: `yoursite/config` in the *user sites directory*.

Note: If anything goes wrong, see the *Installation* (page 498).

Anatomy of a site

A Zotonic site is a folder which lives in the *user sites directory* and contains at least:

- a `config` file: sets the site's hostname and other parameters
- a `sitename.erl` file: initialises the site.

In fact, a site is a special type of *module* (page 39). Like modules, sites usually contain additional resources such as templates and dispatch rules. Unlike modules, however, sites have their own hostname and database connection.

Next steps

- Consult the reference for all site *configuration parameters* (page 491).
- If something goes wrong, consult the *troubleshooting reference* (page 499).

Controllers

Controllers are the Erlang modules which decide what happens when a browser requests a page. Zotonic looks at the *dispatch rules* that match the requested URL, and if a dispatch rule matches, the controller that is named in the dispatch rule is used to handle the request.

Anatomy of a controller

Say we have the following dispatch rule, handling `http://localhost/example`:

```
{example_url, ["example"], controller_example, []},
```

When hitting `/example`, the `controller_example` controller will be initialized and various callback functions on the controller will be called, according to the HTTP protocol flow.

Controllers are pretty self-documenting, thanks to the names of the webmachine callback functions. For instance, when you define a function `resource_exists/2`, it will be called to decide whether or not the page should return a 404 page.

The simplest controller uses Zotonic's `controller_html_helper.hrl` include to serve HTML:

```
-module(controller_example).

-include_lib("controller_html_helper.hrl").

html(Context) ->
    {<<"<h1>Hello</h1>">>, Context}.
```

To return the rendered output of a template file in the module's `templates` directory, use `z_template:render/3`:

```
-module(controller_example).

-include_lib("controller_html_helper.hrl").

html(Context) ->
    % foo and bam will be available as template variables in mytemplate.tpl.
    Html = z_template:render("mytemplate.tpl", [{foo, 'bar'}, {bam, 1234}],␣
↪Context),
    z_context:output(Html, Context).
```

If you need more examples, *mod_base* (page 287) contains many controllers, implementing basic HTTP interaction but also redirects, websockets, et cetera. The *Controllers* (page 344) page lists all available controllers in the Zotonic core.

The Webmachine documentation itself is also very helpful for understanding controllers.

See also:

Create a custom controller (page 197)

Differences between Zotonic's and Basho's Webmachine

Zotonic's fork has been named `webzmachine` and lives in its separate repository at https://github.com/zotonic/webzmachine).

The main differences with Basho's Webmachine are:

- Pluggable dispatch handler
- Support for the HTTP `Upgrade:` header
- Optional caching of controller callbacks results
- Dispatch handler can redirect requests
- Use of process dictionary has been removed
- `webmachine_request` is now a normal (not parametrized) module
- Extra logging

Alltogether, this gave a significant speed boost to Webmachine.

In the specific case of Zotonic the difference was 5 milliseconds (or more) per request (on a 2GHz Core 2 Duo). Without these optimizations we were not able to use Webmachine.

Dispatch rules

Dispatch rules (also known as routes) map URLs to *controllers* (page 11), and the other way around.

A dispatch rule contains a pattern that is matched against the *path* of an incoming request URL. They are also used for the reverse action of generating request URLs in Zotonic.

When an URL is requested by the web browser, the dispatch system looks at that URL and matches it agains all dispatch rules that are loaded. Based on the match, it will call a *controller* (page 344) to handle the request.

Defining dispatch rules

Dispatch rules are defined in a *dispatch* file. The dispatch file must be placed inside the `dispatch/` directory of a module or your site.

A module or site can have multiple dispatch files, and they can have *any* filename. Nota bene: the file may not have the extension ".erl". Otherwise it will be compiled by the Emakefile, which will result in errors as a dispatch file is not a valid Erlang module.

The content of a dispatch file is an Erlang list of dispatch rules.

An example `dispatch` file looks like this:

```
%% Example dispatch rules
[
    {home,       [],                          controller_page,  [ {template, "home.
↪tpl"}, {id, page_home} ]},
    {features,   ["features"],                controller_page,  [ {template,
↪"features.tpl"}, {id, page_features} ]},
    {collection,["collection", id, slug],   controller_page,  [ {template,
↪"collection.tpl"} ]},
    {category,   ["category", id, slug],     controller_page,  [ {template,
↪"category.tpl"} ]},
    {documentation, ["documentation", id, slug], controller_page, [ {template,
↪"documentation.tpl"} ]}
].
```

The module indexer will load all dispatch files. They can be reloaded with the "rescan" button in the admin modules page. Illegal dispatch files are skipped, showing an error message in the Zotonic shell.

When your dispatch rules don't work, check first if there are any typos, then check if your dispatch rules are not overruled by a module that loads earlier. Modules are loaded on priority first, then on module name.

Anatomy of a dispatch rule

A single dispatch rule looks like:

```
{page, ["page", id], controller_page, [{some_option,true}]}
```

Where the elements are:

1. a name identifying the dispatch rule (used by {% *url* (page 456) %})
2. the path matching the request URL's path
3. the name of the controller (*controller_page* (page 357) in this example)
4. a property list with optional arguments to the controller module. Refer to the documentation for respective controller for available options.

See also:

The full list of available *Controllers* (page 344).

Dispatch rule naming

Zotonic extends Basho's Webmachine by allowing (or actually, *requiring*) dispatch rules to be named. The name is the first element of the dispatch rule tuple, and consists of a simple atom. The `z_dispatcher:url_for` function takes a name and creates the URL for it. This allows the developer to not hardcode the URLs everywhere, but use these symbolic names instead.

Dispatch rule names do not have to be unique: if multiple rules with the same name exist, it will look at the first rule that matches the given name and the optional extra arguments that were given.

Say I have these rules:

```
{rulename, ["foo", "bar"], controller_template, [{template, "foo.tpl"}]},
{rulename, ["foo", var], controller_template, [{template, "foo.tpl"}]},
```

Then when I create a URL like this:

```
{% url rulename %}
```

It will match the first rule (rendering the url `/foo/bar`) because no arguments were given. However when I add an argument:

```
{% url rulename var=1 %}
```

It will render the URL `/foo/1`, matching the second dispatch rule and adding the argument in the creation of the URL.

In a template the value of the argument can be retrieved with the *q* variable. In the example where the atom *var* is used:

```
{{ q.var }}
```

Note that any *extra* arguments that are given, are added as query-string parameters:

```
{% url rulename var=1 x="hello" %}
```

Will result in the URL `/foo/1?x=hello`.

URL match pattern

Every element in the URL pattern list matches to a "directory level" in the request URL. In the example, the pattern will match a URL like "page/1234" but not "pages/1234" and also not "page/1234/x".

The possible path elements are:

- Strings: fixed parts that must match with the request url
- atoms: bind to the text at that position
- '*': a special atom binding to the remaining part of the request URL, this must be the last element of the path

URL matching using regular expressions

Some developers need very particular control of dispatch in order for their applications to function as they want them to.

Say you want to only accept numerical arguments as an id in:

```
{foo, ["foo", id], controller_foo, []}
```

The you can use a dispatch rule with a regular expression test:

```
{foo, ["foo", {id, "^[0-9]+$"}], controller_foo, []}
```

or, you can specify http://erldocs.com/R14B02/stdlib/re.html?i=14&search=re:#run/3 some extra options:

```
{foo, ["foo", {id, "1?2?", [notempty]}], controller_foo, []}
```

(In this case, the id must contain a 1 or a 2, amongst any other characters)

URL matching using callback modules

When all else fails, there is another option when you are, really, really, desperate for a specific check. You can call a module:

```
{foo, ["foo", {id, {foo_module, foo_check}}], controller_foo, []}
```

Though note that this is (currently) an extremely expensive operation, because it is called in the `z_sites_dispatcher` gen_server which handles the matching of all incoming requests for all sites in one single process.

When matching against "foo/bar", the module is called as:

```
foo_module:foo_check("bar", Context).
```

Dispatch rule troubleshoooting

Check the Syntax: Load your dispatch file in from the EShell with file:consult/1 and see if it returns errors.

Dispatch Rules are Order-sensitive: Dispatch rules are processed top-to-bottom in the file. Are any rules above your rule capturing the cases you are trying to match. If so, move your rule up, but bear in mind that you don't want to break those rules either.

View which values are passed to the template using tag *debug*:

{% debug %}

URL rewriting

Before URLs are matched, they first can be *rewritten* to match something else. This is a powerful mechanism that allows you do anything you like with URLs.

The URL rewriting mechanism allows one to set extra context variables or change the (internal) URL so different dispatch rules get triggered.

mod_translation (page 317) uses this mechanism to prefix each URL with the language code of the currently selected language.

Todo

document this fully, using mod_translation example

Domain-dependent language selection

An application of URL rewriting allows you to set the Zotonic language based on the domain that is being requested on your site. To set up domain-based language detection using the following code snippet:

```
observe_dispatch_rewrite(#dispatch_rewrite{host=Host}, {Parts, Args}, _Context) ->
    Language = case Host of
                   "example.nl" -> nl;
                   "example.de" -> de;
                   _ -> en  %% default language
               end,
    {Parts, [{z_language, Language}|Args]}.
```

This leaves the request URI intact (the *Parts* variable), but injects the *z_language* variable into the request context, this overriding the language selection.

For this setup to work, this requires you to have the `{redirect, false}` option in your site, and the appropriate `hostalias` directives for each host. See *Anatomy of a site* (page 11) for more details on this.

Unmatched hosts/domains

The dispatcher finds the correct site based on the `Host` in the request. If no site can be found then the dispatcher will first check all enabled sites with a `#dispatch_host` notification to see if any site has a known redirect.

If this fails then the dispatcher will select a default site (usually `zotonic_status`) to handle the request.

If no site is running then a bare bones *404 Not Found* page will be shown.

See *mod_custom_redirect* (page 288) for redirecting unknown domains.

Unmatched paths

If the dispatcher can't find a match a dispatch rule against the request path then it will check the site's modules using a `#dispatch` notification.

The module *mod_base* (page 287) will check the request path against the `page_path` property of all resources. After that the module *mod_custom_redirect* (page 288) will check the configured redirect locations.

Dispatch rule BNF

A dispatch rule is built up as follows:

```
{RuleName, UrlPattern, ControllerModule, ControllerArgs}
RuleName = atom()
PathSpec = [PathSegmentSpec]
PathSegmentSpec = StaticMatch | Wildcard | Variable
StaticMatch = string()
Wildcard = '*'
PathVariable = atom() | {atom(), RegExp} | {atom{}, RegExp, ReOptions}
RegExp = string()
ReOptions = [term()]
ResourceModule = atom()
ResourceArgs = [{Key,Value}]
```

All *PathVariables* in the matching rule are made available to the resource through `z_context`. The *ResourceArgs* proplist is passed to `ControllerModule:init/1`.

PathVariables are part of the request-scope configuration of *ControllerModule* . Things like the ID, name or category of a page being requested can be gathered effectively here. Judicious use of PathVariables can substantially reduce the number of dispatch rules while making them easier to read.

ControllerArgs is the rule-scope configuration of ControllerModule. It makes it possible to reuse a well-designed resource module in many dispatch rules with different needs. ControllerArgs is effective for establishing implementation details like the template to be used, whether or not to do caching and where to load static resources from.

Zotonic dispatch rules are identical to Webmachine's with the addition of RuleName. Webmachine's dispatch rules are described in detail at http://webmachine.basho.com/dispatcher.html .

See also:

mod_custom_redirect (page 288), *mod_base* (page 287)

Resources

Resources are Zotonic's main data unit. You may want to familiarise yourself with the Zotonic *data model* (page 3) in the User Guide.

Resource properties

Resources are very flexible data units: they can have any property that the developer needs them to have. However, by default, Zotonic's admin is designed to edit a common set of properties.

See also:

m_rsc model reference (page 426)

Categories

Every resource belongs to a single category.

There is no real distinction between rsc records that are a person, a news item, a video or something else. The only difference is the *category* of the rsc record, which can easily be changed. Even categories and predicates themselves are represented as rsc records and can, subsequently, have their own page on the web site.

Categories are organized in a hierarchical fashion, and used to organize the resources into meaningful groups. Zotonic has a standard set of categories (see *Domain model* (page 6), but it is very usual to define your own in your own site, resulting in a custom *domain model*.

In the database, categories are stored in an extra metadata table, `category`, which defines the hierarchy of categories using the Nested Set model. The tree is strictly hierarchical: Every category has at most a single parent category, and every resource belongs to exactly one category. That a resource can't belong to more than a single category is done to maintain the datamodel's simplicity and speed of the searches in the system.

Since in Zotonic, *everything is a resource*, categories *themselves* are also resources, namely, resources of the category *category*. This allows the category to be titled and described, just like other resources. The category table only describes the nested hierarchy of the categories. All other properties of a category are defined by its rsc record.

See also:

m_category (page 414) model reference

Medium

Medium management is described in full in *Media* (page 23). Media metadata is stored in a separate table, called `medium`, since one media is a medium. When a resource contains a medium, this table holds a record describing it. Amongst others, it stores its mime type, width, height and file size.

Besides the `medium` table, a `medium_deleted` table exists. When a medium is deleted then any files referenced by that medium will be added to this table. Zotonic periodically checks this table to delete files that are no longer referenced by any media.

See also:

m_media (page 421)

Blocks

Blocks are a specific feature in a resource. The `blocks` property of a resource is a list of blocks which can be dynamically added and removed from the resource in the admin edit page. Each module can define their own blocks, which consist of an edit template and a view template.

The survey module uses the blocks feature to allow you to dynamically create a list of questions which a user has to answer.

Todo

Fix blocks documentation

How resources are stored

Each *resource* on a site is stored in the `rsc` database table. The resource's properties are stored in two ways.

- The core properties are persisted in separate columns. They include id, name, category, modification date, path, publication period. These properties are used for searching, filtering and sorting resources. As they can have unique or foreign key constraints, they help in preserving data sanity.
- All other properties are serialized together into one binary blob column named `props`. This includes any custom properties that you set on the resource. These serialized properties cannot be used for finding or sorting data, but only for later retrieval.

Storing properties in a serialized form is a flexible approach. You can save any property on a resource without having to make changes to your database schema.

Imagine you wish to store whether resources are liked by users. Just update the resource and set a custom `is_liked` property:

```
m_rsc:update(123, [{is_liked, true}], Context).
```

`is_liked=true` will now be stored in the database for resource `123`, so you can retrieve it like you would any other property:

```
?DEBUG(m_rsc:p(123, is_liked, Context)).
%% prints: true
```

This flexible approach is fine for custom properties that you only want to retrieve and display. However, if you need to *find* all liked resources, you need to define `is_liked` as a pivot column (see below).

See also:

m_rsc model reference (page 426)

Pivots

Pivot columns

If you want to search by or order on any custom defined property, you need to define your own database column in a *custom pivot*.

See also:

Custom pivots (page 205)

Note: Zotonic is smart enough that when you enter any textual information into any resource property, it will extract this and put it in the `pivot_tsv` column, for use in full-text searches.

The pivot queue

When the version number or modification date of a resource is updated then its id is added to the *pivot queue*. Zotonic has a pivot process running in the background which looks at this queue and for each queued resource, extract all texts and some other information from the record, filling the pivot columns of the rsc record. The pivot columns are used for searching, they contain amongst others the full text index.

The `rsc_pivot_queue` table is used to hold the queue of resource ids that are waiting to be pivoted.

The `pivot_task_queue` holds a second queue for more generic task processing: it holds references to functions which need to be called in the background.

Identities

An rsc record can become a user by adding the user's credentials to this table. A single user can have multiple kinds of credentials, think of his/her username, openid uri etc. A user isn't necessarily a person.

See also:

m_identity (page 420).

Deleted resources

Whenever a resource is deleted, an entry is added to the `rsc_gone` table. The page and id controllers will serve a *410 Gone* when a deleted resource is requested.

See also:

m_rsc_gone (page 431).

Templates

Zotonic's template syntax is very similar to the Django Template Language (DTL).

The templates in Zotonic are based on the Django Template Language (DTL), using a customized version of the excellent ErlyDTL library. Over the years, Zotonic's version of ErlyDTL has diverged, adding Zotonic-specific features and more powerful expression possibilities. However, the main template syntax remains the same:

The double accolade/brace construct outputs the value of the variable:

```
{{ foo }}
```

Optionally, you can pipe these variables through a so-called filter, which is applied before output:

```
{{ foo|lower }}
```

Template tags allow you to express complex constructs like loops and branches:

```
{% if username == 'Arjan' %} Hi, Arjan {% endif %}
```

Template locations and the lookup system

All templates are located in the `templates` directory of modules or sites. Templates are referred to by their full filename. When a template is inside a directory then the full path of the template must be given.

For example, say we have two templates:

mod_example/templates/foobar.tpl
mod_example/templates/email/email_base.tpl

The above are referred to as `foobar.tpl` and `email/email_base.tpl`; just `email_base.tpl` will not find the email template.

All templates of all modules are grouped together, regardless of which module they are defined in. The module name is never given as part of the template name.

Templates for pages

When showing a page, Zotonic looks up templates in order of specificity and renders the first template it finds:

1. `page.name.tpl` (unique name)
2. `page.category.tpl` (category)

3. `page.tpl` (fallback)

So if you have a page in the category 'text' and that page has a *unique name* (page 426) 'my_text_page', Zotonic will look for the following templates:

1. `page.my_text_page.tpl` (unique name)
2. `page.text.tpl` (category)
3. `page.tpl` (fallback)

See also:

- *controller_page* (page 357), *catinclude* (page 444).

Module priority and overriding templates

Templates with the same filename can be defined in multiple modules. The actual template which is selected depends on the priority of the module.

The *module priority* is a number defined in the module's code and is usually a number between 1 and 1000. A lower number gives a higher priority. Templates in a module with higher priority hide templates in a module with lower priority.

When two modules have the same priority then the modules are sorted by their name. That means that, given the same priority number, `mod_aloha` has higher priority than `mod_hello`.

This mechanism allows any module (or site) to replace every single template in the system with its own version.

Note: Including all similar named templates

The tag `{% all include "foobar.tpl" %}` will include all templates named `foobar.tpl`.
Where the tag `{% include "foobar.tpl" %}` only includes the highest priority `foobar.tpl`.

See also *all include* (page 441) and *all catinclude* (page 441).

Module priority and overriding lib files

Exactly the same module priority is also valid for all files in the `lib` directory of modules.

This allows any module to change the static css, javascript, images, favicon.ico, robots.txt and other static files with its own version.

User Agent selection

The module priority is a very powerful mechanism for extending and adapting Zotonic.

But what if a page requested with a mobile phone should be served with a different template than the same page requested with a desktop computer?

For this there is another template selection mechanism, based on the categorization of the device requesting the page.

User agent classes

Every request Zotonic classifies the device using the *User-Agent* request header. The possible classifications are:

text Screen readers, feature phones, text only browsers.

phone Smart phones, capable of javascript and having a touch interface or other pointing device.

tablet Big screen, javascript, modern browser and touch interface.

desktop Big screen, javascript, modern browser and pointing device.

The selected class is available in `m.req.ua_class` or from Erlang `z_user_agent:get_class/1`.

Note: More properties can be found using `m.req.ua_props` or `z_user_agent:get_props/1`.

The four user agent classes map to subdirectories of the `templates` directory:

mod_example/templates/desktop/...
mod_example/templates/phone/...
mod_example/templates/tablet/...
mod_example/templates/text/...

All templates that are not in those sub-directories are categorized as *generic*.

Lookup by user agent class

The template system follows a strict hierarchy between the different user agent classes:

desktop → tablet → phone → text → generic

Where the system starts looking from the current user agent class to the right. So for a phone, the templates in the `tablet` and `desktop` directories will never be considered.

Combination of user agent and module priority

The user agent class and the module priority are two dimensions of the template selection process.

The module priority is more important than the user agent class.

A mismatch in user agent class (e.g. a desktop template when looking for a phone version) will never be selected. A sub-optimal version (e.g. a generic or text version instead of a phone version) will be selected if that sub-optimal version resides in a module with higher priority than the module with the better matching version.

The *all include* tag will select the best version from all modules. Again skipping any user agent mismatches.

Note: Building templates and mobile first.

The lookup strategy for templates conforms to a *mobile first* strategy. When adding a page or building a site, the idea is to start with the simplest, text only, version of the site. The text only version is then placed in the `templates/text` directory. Next will be adding more features, markup and interaction for the phone version. Only then moving up to the big screen for tablet (touch) or desktop (mouse).

Note: Seeing which template is selected.

mod_development implements a screen where it is possible to see in real time which templates are included and compiled. The full path of all templates can be seen, giving insight in the template selection process.

See also *mod_development* (page 289)

Template variables

Global variables

The following properties are always available in a template.

zotonic_dispatch The name of the dispatch rule that was applied to render the current page.

zotonic_dispatch_path A list containing the request path used as initial input for the dispatcher. The path is split on `/` and after an optional rewrite. This means that the list doesn't contain the language prefix. For example, the path `/en/foo/bar?a=b` will give the list `["foo", "bar"]`.

zotonic_dispatch_path_rewrite Same as zotonic_dispatch_path, but set to the path after an optional internal request rewrite inside the dispatcher. For example if a resource has its *page_path* set to `/foo` and the requested path is `/en/foo` then the `zotonic_dispatch_path` will be set to `["foo"]` and the `zotonic_dispatch_path_rewrite` could be set to something like `["page", "1234", "foo-slug"]`.

z_language The currently selected language. This an atom, for example: `en`.

q A dictionary containing the current request's query variables. For GET requests, these are the arguments passed from the query string (e.g. `?foo=bar`); for POST requests, these are the values posted in the POST form. For more access to the raw request data, look at the *m_req* (page 424) model.

now The local date and time in Erlang tuple notation, for instance `{{2014,4,17},{13,50,2}}`.

m `m` is not really a value, but it's an indicator to trigger a lookup in one of Zotonic's *Models* (page 411). For instance the *m_rsc* (page 426) model is always exposed and can be used like this `{{ m.rsc[123].title }}`.

z_trigger_id Only available in postback contexts. The id of the html element triggering a postback.

z_target_id Only available in postback contexts. The id of the html element that is the target of a postback.

z_delegate Only available in postback contexts. The name of the Erlang module handling the postback event.

Besides these variables, all key/value pairs that are set in the `#context{}` record (using `z_context:set/2`) that was used to render the current template are also exposed into the template's global scope.

Debugging

To print the variables in your template for debugging, you can use the scomp-debug tag:

```
{% debug %}
```

Tags

Tags add logic and flexibility to your templates. The general syntax for a tag is the following:

```
{% tagname param1=value param2=value %}
```

Some tags are *block tags* and therefore consist of a start and an end tag. The name of the end tag is always `end` plus the name of the opening tag:

```
{% tag %}
    ...
{% endtag %}
```

For instance, use the `for` tag to loop over lists:

```
{% for article in articles %}
    {{ article.title }}
{% endfor %}
```

And the `if` tag to check conditions:

```
{% if article.is_published %}
    There you go: {{ article.title }}
{% else %}
    Sorry, the article hasn't been published yet!
{% endif %}
```

See also:

- List of *all tags* (page 440) reference.
- *Create your own tags cookbook* (page 205).

Filters

Filters are used to modify values you want to show or use in your templates. For example:

```
{{ value|lower }}
```

will lowercase the input value using the *lower* (page 403) filter.

See also:

a listing of all *filters* (page 366).

Models

A template model provides data to a template through the syntax: `m.modelname.property`. For example:

```
{# Get the site's title #}
{{ m.site.title }}

{# Fetch the title of the page with name page_home #}
{{ m.rsc.page_home.title }}

{# Fetch the title of the page whose id is the integer 1 #}
{{ m.rsc[1].title }}

{# Fetch the title of the page whose id is the template variable id #}
{{ m.rsc[id].title }}

{# Perform a search on all persons #}
{% for p in m.search[{query cat='person'}] %}{{ p.title }}{% endfor %}
```

See also:

- list of *all models* (page 411) in the reference
- *Create a custom model* (page 199) cookbook

Media

To include a resource's depiction, use *image* (page 449):

```
{% image id %}
```

You can pass extra parameters to adjust the image on the fly:

```
{% image id width=200 height=200 crop %}
```

The image will then be resized and cropped to the specified 200x200 pixels.

See also:

image (page 449) for all parameters

Media classes

Instead of inline image tag parameters, you can use media classes to define image transformations. The advantage is that this image definition can then be reused amongst templates.

Create a `templates/mediaclass.config` file in your site directory:

```
[
    {"thumb", [
        {width, 200},
        {height, 200},
        crop
    ]}
].
```

This defines a media class called 'thumb', which can be used to display a 120x120 cropped square image. You then only need to refer to this media class in your image tag:

```
{% image id mediaclass="thumb" %}
```

The image URL will have a checksum embedded in it so that when the contents of the media class is changed, all images which use that media class will be regenerated to reflect the new media class.

Raw ImageMagick options

Besides the normal image processing options, as described in *image* (page 449), it is possible to add literal ImageMagick convert commands to the mediaclass definition.

For example:

```
{magick, "-level 90%,100% +level-colors \\#FE7D18,\\#331575"}
```

(Note that you have to double any backslashes that were needed for the `convert` command line.)

This command is given *as-is* to the ImageMagick *convert* command, therefore it is best to first try it with the command-line *convert* command to find the correct options and command line escapes needed.

There are three variations: `pre_magick`, `magick`, and `post_magick`. The only difference is that the `pre_magick` is added before any other filter argument, `magick` somewhere between, and *post_magick* after the last filter.

In this way it is possible to pre- or post-process an image before or after resizing.

See http://www.imagemagick.org/Usage/ for examples of using ImageMagick from the command line.

User-agent specific images

Since `mediaclass.config` files are found using the *Template locations and the lookup system* (page 19), it is subject to the same selection rules that normal templates fall under.

The consequence is that you can have multiple `mediaclass.config` files, for instance one in *desktop/*, one in *phone/*. The media classes defined in those subdirectories can have the same names. This way you can make thumbnail sizes smaller for phones, or serve higher-quality JPEG file for desktop browsers.

See *User Agent selection* (page 20) for the details on the user-agent selection mechanism.

Actions

The action defines what should happen when the wire is triggered. Actions can be client-side (such as JavaScript animations) or server-side postbacks.

Trigger actions from JavaScript

To trigger an action from an HTML element, you attach a wire to the element:

```
<a href="#" id="link">Click me!</a>
{% wire type="click" id="link" action={fade_out target="link"} %}
```

The wire's `id` value must match the `id` value of the HTML element. This wires up a link with a action-fade_out action, so that when the link is clicked, it fades away.

Actions can be called from the template, but can also be called when some server-side event occurs.

See also:

Auto-generated identifiers (page 26), *Create a custom action* (page 197)

Server postbacks

Postbacks are server-side actions. For instance, to submit a form asynchronously through Ajax, use a postback:

```
{% wire type="submit" id="myform" postback="form_submitted" delegate="mysite" %}
<form id="myform" method="post" action="postback">
    <input name="username" />
    <button>Submit form</button>
</form>
```

This will submit the form over Ajax; the result is that a function will be called in the specified delegate module `mysite.erl`, called `event/2`:

```
event(#submit{}, Context) ->
    io:format("The value of 'username' is: ~s~n", z_context:get("username",␣
↪Context),
    Context.
```

See also:

postback reference

Trigger browser actions from the server

See also:

listing of all *actions* (page 320).

Named actions

If you want to trigger actions from your JavaScript code, give the action a name:

```
{% wire name="my_action" action={growl text="Hello World"} %}
```

You can then refer to it in your JavaScript code:

```
z_event("my_action");
```

And pass arguments to the action:

```
z_event("my_action", { foo: bar });
```

The argument `foo` will become a query argument, that you can access in your Erlang module with `z_context:get_q(foo, Context)`.

Adding CSS and JavaScript

JavaScript

Auto-generated identifiers

If you include a template many times (i.e. from a for loop), then having fixed element identifiers are no good. Zotonic provides a mechanism to generate an identifer which has a unique value within the template.

To prefix the id with a unique value (per invocation of the template) prefix the id with a #-sign:

```
<div id="{{ #foo }}">
```

This special notation will replace `#foo` with an auto-generated identifer, which will expand to something like this:

```
<div id="ubifgt-foo">
```

Unique ids can also be generated inside a `for` loop:

```
{% for id in mylist %}
    <li id="{{ #foo.id }}">{{ id.title }}</li>
{% endfor %}
```

This will generate HTML like this:

```
<li id="gdjqa-foo-1234">Some great news</li>
```

When using a scomp-wire tag, that same unique id can be referenced:

```
{% for id in mylist %}
    <li><a id="{{ #list.id }}" href="#">{{ m.rsc[id].title }}</a></li>
    {% wire id=#list.id action=some_action %}
{% endfor %}
```

Icons in templates

Zotonic provides a couple of ways to show icons in templates:

- *mod_artwork* (page 285) gives access to FontAwesome and Material Design icons. It also has a number of other icon collections, mostly PNG images. Activate the module and follow the instructions on the doc page.

- Zotonic icons provided by *mod_base*. This is explained on the current page.

To create a certain amount of consistency across modules, Zotonic comes with a small set of commonly used icons and CSS classes (edit, help, close, etcetera) plus the Zotonic logo.

Use cases:

- You create your frontend from scratch, but you also have pages in your site that are provided by other modules, for instance the login screens. It would be good if the social login icons show up.
- You are writing a template or module and like to take advantage of ready available icons.
- You are writing frontend styles in LESS and you would like to extend Zotonic / FontAwesome / Material Design icons.

Include the Zotonic icons CSS file in your template:

```
{% lib
    "css/z.icons.css"
%}
```

Then use this syntax in your template HTML:

```
z-icon z-icon-<name>
```

For instance:

```
<span class="z-icon z-icon-off"></span>
```

See also:

Icons (page 482) reference

Forms and validation

Validators for HTML form fields.

Validators check if form fields have an acceptable value. They check *both client side and server side* if the input fields are valid.

Validators check the input element with JavaScript and prevent posting the form unless the validation is passed. All validations are also done on the server. This prevents people bypassing the validation checks in their browser. When the validation does not pass on the server side then the post will fail.

Validated form fields are available to Erlang code using the `z_context:get_q_validated/2` function.

When an input field has been verified then it is available to Erlang programs via the function `z_context:get_q_validated/2`.

To check if two fields are equal:

```
<input type="password" id="password" name="password" value="" />
<input type="password" id="password2" name="password2" value="" />
{% validate id="password" type={confirmation match="password2"} %}
```

The password field is now available to the Erlang code with:

```
Password = z_context:get_q_validated("password", Context).
```

See also:

- listing of *all validators* (page 476)
- the scomp-validate tag.

Search

Using the query search API you can retrieve lists of resources in various ways. In your templates, you do so through the *search model* (page 432):

```
{% for id in m.search[{query (options go here...) }] %}
```

For instance, to select all news items, ordered by their modification date, newest first:

```
{% for id in m.search[{query cat='news' sort='-rsc.modified'}] %}
    {{ id }}
{% endfor %}
```

Trying it out

Of course you can create your own `for`-loop in a template, but there are easier ways to check out the inner workings of the query model: through your browser.

The query-model is exposed to the browser in (currently) 2 URLs: the Atom feed module for creating a customized update feed, and the API for receiving lists of ids in JSON.

Get all resource of the "documentation" category on zotonic.com:

http://zotonic.com/api/search?cat=documentation

Get a feed of most recent documentation containing the word "filter":

http://zotonic.com/feed/search?cat=documentation&text=filter

Note: `mod_atom_feed` automatically sorts on last-modified date, `api/search` doesn't.

Query arguments

authoritative

Boolean, filters whether a resource is considered authoritative (belonging on this site) or not:

```
authoritative=1
```

cat

Filter resources on a specific category:

```
cat='news'
```

Specifying multiple 'cat' arguments will do an OR on the categories. So to select both news and person resources:

```
cat='news' cat='person'
```

cat_exact

Filter resources to include the given category, but exclude any subcategory:

```
cat_exact='news'
```

cat_exclude

Filter resources to exclude the given category:

```
cat_exclude='meta'
```

id_exclude

Filter resources to exclude the ones with the given ids:

```
id_exclude=123
```

filter

Filtering on columns:

```
filter=['pivot_title', 'Hello']
```

In its most simple form, this does an 'equals' compare filter. The `filter` keywords expects a list. If the list is two elements long, we expect the first column to be the filter column name from the database table, and the second column name to be the filter value:

```
filter=['numeric_value', `gt`, 10]
```

If the filter is a three-column list, the second column is the operator. This must be an atom (surround it in backquotes!) and must be one of the following: `eq`, `ne`, `gt`, `gte`, `lt`, `lte`; or one of =, <>, >, >=, <, <=:

```
filter=['numeric_value', `>`, 10]
```

It is possible to define an OR query for multiple terms:

```
filter=[ ['numeric_value', `>`, 10], ['numeric_value', `<=`, 0] ]
```

hassubject

Select all resources that have an *incoming edge* from the given page, which is specified by the argument (the page id `123` in the example, or the unique page name `tag_gift`). Optionally, you can pass the name of a predicate as the second argument, to specify that the connection should have this predicate.

So, to select all resources that have an incoming edge from a subject with id `123`:

```
hassubject=123
```

Alternatively, use the subject's unique name:

```
hassubject='tag_gift'
```

Specifying this multiple times does an AND of the conditions:

```
hassubject=123
hassubject=[123,'author']
```

hasobject

Like `hassubject`, but selects all pages that have an **outgoing edge** to the given page, which is specified by the argument. Optionally, you can pass the name of a predicate as the second argument, to specify that the connection should have this predicate:

```
hasobject=123
hasobject='tag_gift'
hasobject=[123,'hasdocument']
```

hasanyobject

Like `hasobject`, but allows to define an OR operation on the edge. You can define multiple combinations of predicates and objects; any resource having such an outgoing edge will be matched. The argument is a list. Each element in the list is either an id or an id/predicate combination.

To select all resources that have an outgoing edge to an object with id 1, 2 or 3:

```
hasanyobject=[1, 2, 3]
```

For each list element, you can add the connection's predicate. So, to select all resources that have an outgoing 'author' edge to an object with id 123:

```
hasanyobject=[[123, 'author']]
```

And to do the same but also include resources that have an 'editor' edge to an object with id 456:

```
hasanyobject=[[123, 'author'], [456, 'editor']]
```

Substitute `'*'` for the object id to match *any* object. So, to select all resources that have any author or editor edge:

```
hasanyobject[['*', 'author'], ['*', 'editor']]
```

You can also mix the two types of elements. To select all resources that have an author or a connection (with any predicate) to resource 2 or 3:

```
hasanyobject[['*', 'author'], 2, 3]
```

match_objects

Find the resources that have similar object edges as the given resource. This is done using a full text query. The resource with most overlapping objects ids will be returned first:

```
match_objects=1234
```

An `id_exlude=...` is automatically added for the resource in the argument.

is_featured

A boolean option that specifies if a page should be featured or not:

```
is_featured
```

is_published

Select published, unpublished or omit the publish check. Legal values are true, false or all:

```
is_published='all'
```

is_public

Filter on whether an item is publicly visible or not. Valid values are 'true', 'false', 'all':

```
is_public='false'
```

upcoming

Specifying 'upcoming' means that you only want to select things that have a start date which lies in the future. Like the name says, useful to select upcoming events:

```
upcoming
```

ongoing

Specifying 'ongoing' means that you only want to select things that are happening now: that have a start date which lies in the past, and an end date which lies in the future:

```
ongoing
```

finished

Specifying 'finished' means that you only want to select things that have a start date which lies in the past:

```
finished
```

unfinished

Specifying 'unfinished' means that you only want to select things that have an end date which lies in the future:

```
unfinished
```

unfinished_or_nodate

Specifying 'unfinished_or_nodate' means that you only want to select things that have an end date which lies in the future or no start date:

```
unfinished_or_nodate
```

sort

Sort the result on a field. The name of the field is a string which directly refers to the SQL join that is being used. If you specify a dash (−) in front of the field, the order is descending. Leaving this out or specifying a + means ascending.

Some sort fields:

- `rsc.modified` - date of last modification
- `rsc.pivot_date_start` - the start date specified in the admin
- `rsc.pivot_date_end` - the end date specified in the admin
- `rsc.pivot_title` - the title of the page. For multilingual sites, the behavior of sorting on title is undefined.
- `seq` - sequence number of the first edge (ignored if no edge is joined)
- `edge.created` - creation date of the first edge (ignored if no edge is joined)

For all the sort fields, you will have to consult Zotonic's data model. Example sorting on modification date, newest first:

```
sort='-rsc.modified'
```

custompivot

Add a join on the given custom pivot table. The table is joined to the primary `rsc` table: `custompivot=foo` (joins the `pivot_foo` table into the query)

The pivot tables are aliassed with a number in order of their occurrence, with the first pivot table aliassed as `pivot1`. This allows you to do filtering on custom fields like this:

```
{query custompivot="pivotname" filter=["pivot1.fieldname", `=`, "hello"]}
```

hasobjectpredicate

Filter on all things which have any outgoing edge with the given predicate:

```
hasobjectpredicate='hasdocument'
```

hassubjectpredicate

Filter on all things which have any incoming edge with the given predicate:

```
hassubjectpredicate='author'
```

text

Perform a fulltext search on the primary "rsc" table. The result will automatically be ordered on the relevancy (rank) of the result:

```
text="test"
```

Use prefix `id:` to find specific resources by id or name:

```
text="id:1000"

text="id:1000,1001,1002"

text="id:category,1"
```

query_id

Load the query arguments from the saved `query` resource:

`query_id=331`

See also:

Query resources (page 34)

qargs

Take all the arguments from the current request and use these. The arguments have to start with a `q`, for example:

```
http://example.com/search?qs=test&qcat=text
```

With the query:

```
m.search.paged[{query qargs page=q.page pagelen=20}]
```

Will find all pages containing the string *test* in the category *text*.

As `qs` is the usual text search argument in forms it is mapped to `text`. All other arguments have the `q` removed and should map to known query-model arguments.

publication_month

Filter on month of publication date

`publication_month=9`

publication_year

Filter on year of publication date

`publication_year=2012`

date_start_after

Select items with a start date greater than given value

`date_start_after="2010-01-15"`

It also possible to use relative times:

- `now`
- `+0 sunday` (last sunday or the current sunday)
- `+0 monday` (last monday or the current monday)
- `+1 minute`
- `+1 hour`

- +1 day
- +1 week
- +1 month
- +1 year

Negative offsets are allowed as well. There //must// be a + or − sign.

date_start_before

Select items with a start date smaller than given value:

```
date_start_before="2010-01-15"
```

date_start_year

Select items with an "event start date" in the given year:

```
date_start_year=2012
```

date_end_after

Select items with a end date greater than given value:

```
date_end_after="2010-01-15"
```

date_end_before

Select items with a end date smaller than given value:

```
date_end_before="2010-01-15"
```

date_end_year

Select items with an "event end date" in the given year:

```
date_end_year=2012
```

content_group

Select items which are member of the given content group (or one of its children):

```
content_group=public
```

name

Find resource with a matching unique name. A wildcard can be defined, for example:

```
name=page_*
```

Searching on an empty name or just `*` will return all resources with a defined name. The given name will be trimmed and converted to lowercase before searching.

Filter behaviour

All of the filters work as AND filter. The only exception to this is the `cat=` filter: if you specify multiple categories, those categories are "OR"'ed together, to allow to search in multiple distinct categories with a single search query.

Query resources

Query resources are, as the name implies, *Resources* (page 16) of the special category *query*. In the admin this category is called "search query". it is basically a stored (and thus content manageable) search query. You create an editable search query in an admin page that then is invoked from a template.

When creating such a resource in the page, you will see on the admin edit page an extra text field in which you can add search terms. Each search term goes on its own line, and the possible search terms are equal to the ones described on this page (the *Query-model arguments*).

Translation

Many sites need to support content and templates in multiple languages. Luckily, Zotonic is completely multilingual, out of the box. *mod_translation* (page 317) does all the work for you.

Zotonic will try to serve the page in the language of the user (based on the browser's HTTP Accept header) or the selected language from the URL. If this fails, Zotonic falls back to the default language as configured on the Translation admin page.

In Zotonic, two kinds of information are being translated.

- Static translations: The fixed texts in your templates, like button labels, confirmation texts, etc.
- Translated content, as edited in the admin. The text fields of each *resource* can be translated in multiple languages.

Enabling multiple languages

Before you can use multiple languages, you first need to enable *mod_translation* (page 317).

Next, you need to tell Zotonic which languages you are going to use. You can do this in the /admin, following the *Translation* menu item in the *Structure* submenu.

On this page you can add and remove languages, enable/disable languages and more. Note that a language is always identified by its two letter ISO639-1 language code.

Translation sources

Texts in templates are made localizable with different forms of underscore syntax.

Translate tag:

```
{_ Translate me _}
```

Extended translate tag:

```
{% _ "Example" nl="Voorbeeld" fr="Exemple" %}
```

As part of a tag parameter:

```
{% button text=_"Click me" %}
```

Texts in `erl` files use the `?__()` syntax:

```
?__("Translate me", Context)
```

and using string concatenation:

```
[?__("Edit:", Context), " " ++ Title]
```

Generation of translations

The fixed texts in a Zotonic website are translated using the GNU gettext `.po` file format and its related tools.

In Zotonic, static translations are organized in each Zotonic module. It is the module's responsibility to provide translations for all the texts that it uses in its templates. All files related to static translations live inside the `translations/` subdirectory of a module (remember: a Zotonic site is just a module!).

In the `translations/` directory of the modules you can find the `.po` files containing the translations. They are marked with the their two letter language code. (Optionally you can name your file like: nl.foobar.po as Zotonic will only look at the part till the first '.' for the language code):

```
translations/
- nl.po
- template
|    - mod_foo.pot
- tr.po
```

Here you see that this module, `mod_foo`, has been translated into Dutch (*nl*) and Turkish (*tr*).

The translations/ directory contains a subdirectory which contains `mod_foo.pot`, which is the translation *template*, on which new translations will be based.

The basis for these files (the translation *template*), is the `.pot` file which is located in the `template/` subdirectory of the translations directory. This `.pot` file is regenerated when you click on the 'Generate .pot files' button on the Translation page in the admin. Alternatively, from your Zotonic shell:

```
mod_translation:generate(Context).
```

Zotonic will parse all your templates and Erlang modules for translatable strings. These strings are then added to the .pot file.

Creating a new translation for a module

First, add a language in the admin with the 2-letter code for that language.

Say, we're adding Polish, `pl`. Now copy the `.pot` template file to the 2-letter code `.po` file:

```
$ cd modules/mod_foo
$ cp translations/template/mod_foo.pot translations/pl.po
```

Now, open `pl.po` in your favorite editor and start translating the strings. A good po file editor can be found at: http://www.poedit.net/

Updating translation strings

When templates change, often the translatable strings change: more strings are added, strings are changed, or removed. When this happens, the translation files need to be kept in sync.

The *Translation status* page in the admin gives an overview, per module / language combination, of the amount of strings that are translated for each language.

After pressing the *Generate .pot files* button in the translation admin the `.pot` files are updated, but the existing `.po` files are not.

GNU gettext comes with a tool, `msgmerge`, which looks at the changed strings in a `.pot` file and changes the translated strings in a language's `.po` file accordingly:

```
$ cd modules/mod_foo/translations
$ msgmerge -U -N nl.po template/mod_foo.pot
```

This will merge the new strings into the existing `nl.po` file.

To update all `.po` files in the directory:

```
$ cd modules/mod_foo/translations
$ find . -name "*.po" -print0 | xargs -0 -I file msgmerge -U -N file template/*.pot
```

After doing this, you'll need to edit each `po` file again to check if there are new strings which need translation, edit existing strings, etc.

Helpful commands

To remove duplicates (and make a backup first), use:

```
$ cat nl.po > nl~.po && msguniq nl.po -o nl.po
```

Translated content

When you have enabled languages on the Translation page of the admin you will see a Translations item on the right of the edit page.

Each language has a checkbox next to it: When you click a checkbox, the language will become visible as a tab on your content items.

Resources in Zotonic are translated on a per-page basis. This allows you to start translating your site by translating the most important pages first.

Text searches, translations and stemming

For text searches a full text index is maintained. This full text index is stemmed according to the site's configured default language.

All translations are added to the same full text index. This combined text is stemmed using a single stemmer. The selected stemmer depends on the configured default language (config key `i18n.language`). The stemmer can be overruled by setting the config key `i18n.language_stemmer` to the two letter iso code of the language matching with the stemmer. You have to make sure that the stemmer is configured in PostgreSQL otherwise the pivot process will crash with a SQL error.

Access control

Access control is about defining who is allowed to access certain resources. It takes two steps:

1. The user is *authenticated*, that is, the user's identity is determined based on some form of identification in the request, such as a session cookie.
2. After the user has been identified, we can decide whether that user is *authorized* to access a certain URL or resource.

Authentication

For each HTTP request that Zotonic receives, it looks for some form of credentials that can identity the user. This can be a username/password combination when the user logs in for the first time, and a *session cookie* for subsequent requests. When Zotonic finds some credentials, it checks them for validity. If the credentials are valid, the user is said to be authenticated and authorization can start.

The first request, that does not yet have a session cookie or whose session cookie has expired, needs to contain some credentials in order to be authenticated. The *logon controller* (page 356) takes care of processing login requests and checks for the presence of a 'remember me' cookie for automatic login. It then responds with a fresh session cookie that the client will send along with subsequent requests.

Authenticating subsequent requests, that have a session cookie, does not take place until a session is ensured (or continued) for the request *context*. This is commonly done by the *controller* handling the request by a call to `z_context:ensure_all/1`.

Notifications

Zotonic relies on a number of *notifications* (page 46) to perform authentication. You can observe any of these notifications to customise Zotonic's authentication behaviour.

Notification	Type	Return	Description
auth_confirm	foldl	Context	Sent when a user id has been confirmed.
auth_confirm_done	notify		
auth_logon	foldl	Context	Sent when a user has been authenticated.
auth_logon_done	notify		
auth_logoff	foldl	Context	Sent when a user is about to log out, removing the authentication from the current session.
auth_logoff_done	notify		Sent when a user has been logged out.
auth_autologon	first	{ok, UserId}	Sent for new sessions from `z_auth:logon_from_session/1`. Will attempt to authenticate the session as UserId (if there was any observer responding to the notification).
#user_is_enabled{id}	first	boolean()	Ask observers if the user is enabled (allowed to login, to be authenticated). If the result is `undefined`, the resource `is_published`, `publication_start` and `publication_end` is checked instead.

Authorization

Once the request has been authenticated, authorization is initialized by sending an `#acl_logon{}` notification. Should the session get logged out, loosing its authentication, the authorization is cleared by sending a `#acl_logoff{}` notification.

Once authorization has been initialized for a request *context*, operations against objects can be checked by the `z_acl` module from Erlang code, and by the *m_acl* (page 411) model from *templates* (page 19).

Protecting access to controllers

The first point of contact for authorization is in the *controller*'s `is_authorized/2` function. Both *controller_page* (page 357) and *controller_template* (page 362) check for *ACL options* (page 358) in the *dispatch rule* that matched the current request.

See also:

ACL options (page 358)

Protecting access to resources and modules

Zotonic ships with *mod_acl_user_groups* (page 271), a powerful user group-based authorization module. With this module you can define access control rules that determine which user groups are allowed to access which groups of content.

If you only have a handful of users that should all be able to access the admin, have a look at *mod_acl_adminonly* (page 271).

See also:

mod_acl_user_groups (page 271), *mod_acl_adminonly* (page 271)

Custom authorization

No matter what authorization module you use, you can always override Zotonic's behaviour by observing the authorization notifications. This is especially useful if your application has some authorization logic that is not easily expressed in ACL rules.

See also:

acl_is_allowed (page 486)

Notifications

The authorization system sends several notifications that you can hook into to allow or deny user access to specific resources.

Notification	Type	Return	Description
#acl_is_allowed{action, object}	first	boolean()	Check if user is authorized to perform operation on object. Default is `false`.
#acl_is_allowed_prop{action, object, prop}	first	boolean()	Check if user is authorized to perform operation on property of object. Default is `true`.
#acl_rsc_update_check{id}, Props, Props	fold	Props'	Filter properties about to be updated for a resource.
#acl_can_see{}	first	integer()	Get max *visible_for* (page 426) that the user can see.
#acl_logon{id}	first	Context	Initialize context with the access policy for the user.
#acl_logoff{}	first	Context	Clear the associated access policy for the context.
#acl_context_authenticated{}	first	Context	Set the context to a typical user's permissions, do not change the context if an user is logged on. Used by (for example) `m.acl.authenticated.insert.article`

Modules

Modules are the building blocks of Zotonic.

Examples of modules are the `/admin` site, Atom feeds, the `sitemap.xml`, video embed code handling and SEO optimization. Modules also augment the functionality of other modules by adding extra templates and accompanying logic or adding handlers for internal Zotonic events. Good examples are the modules extending the *mod_admin* (page 275).

A module is a directory containing the module's Erlang code, *templates* (page 19), *controllers* (page 11), *dispatch rules* (page 12) and more, all contained in a single module directory tree.

See also:

listing of all *Modules* (page 271).

Looking for more modules?

Check out the Zotonic Module Index:, an index with additional user-contributed modules which are not part of the core Zotonic distribution.

Structure

A module groups related functions together into a single directory. It contains an Erlang module (from here on called the 'module file') and subdirectories for templates, actions, scomps, dispatch rules and more.

The generic structure is:

```
mod_example/
    mod_example.erl
    templates/
    actions/
    etcetera.../
```

The module file

The name of the module file is an Erlang file that *must* be the same as the name of the module's directory. Zotonic scans this file for metadata about the module and uses it to start the module.

The code of the smallest possible module is below:

```
-module(mod_example).
-author("Nomen Nescio <nomen@example.com>").
-mod_title("Your module title").
-mod_description("Description what this module does.").
-mod_prio(500).
```

In this case, the module code only consists of some metadata properties, there is no real code in there. This is fine for a lot of modules: since Zotonic already provides so many functions, there is often little need to write custom code.

The `mod_title` and `mod_description` properties describe your module in natural language: these properties will be visible on the admin modules page. The `mod_prio` property defines the priority of the module. The highest *Priority* (page 43) is 1, the default is 500. Modules with higher priority are checked first for templates, actions, custom tags, etc. Modules with the same priority are sorted by ascending module name.

In cases where you need to execute code when the module starts, you can export an optional `init/1` function. The parameter is a context record initialized for the site the module will be running in. This is useful when you need to initialize the database or other data structures for which you don't need a running process. When you also need to execute code when a module stops you can export an optional `terminate/2` function. This function will be called when the module terminates. The first parameter is a Reason parameter which indicates why the module stopped. The second a context record similar to the one in the `init/1` function.

When you do need a running process, read about those in the next topic, *gen_server based modules* (page 45).

Module subdirectories

Besides the module code file, a module usually has one or more subdirectories. These are specially named; different parts of Zotonic scan through different folders.

This section describes what each of the module folders hold.

actions/

This directory holds the *actions* (page 25) defined by the module. Every action name must be prefixed with the word "action" and the module name (without the *mod_*). For example the filename for the action `dialog_open` in the module `mod_base` will be `action_base_dialog_open.erl`

See also:

Actions (page 25)

dispatch/

This directory contains files with *dispatch rules* (page 12). You can name your files however you want, just don't give them the extension `.erl`, because then the Makefile will try to compile them.

See also:

Dispatch rules (page 12)

lib/

The *lib* (short for *library*) directory contains static images, css and javascript files. These files will be served with via the *lib* (page 453) tag using the *lib* dispatch rule. The usual layout of the lib directory is:

```
lib/css/
lib/images/
lib/js/
lib/misc/
```

See also:

the *lib* (page 453) template tag.

scomps/

Any custom tags that you define yourself go into the `scomps/` directory.

Scomps are prefixed in the same way as actions, except that the word "scomp" is used. For example the scomp `button` in the module `mod_base` has as file name `scomp_base_button.erl`.

See also:

Tags (page 22)

controllers/

This directory contains Erlang modules which define controllers which are called from the dispatch system to handle incoming HTTP requests.

Controllers must have unique names, as they are compiled and loaded in the Erlang system. The convention is to prefix every controller with `controller_` and the name of the module, for example `controller_admin_edit.erl`.

See also:

Controllers (page 11)

models/

This directory contains Erlang modules, each of which is a *model* (page 23).

The module name of a model always starts with `m_`, for example `m_comment`. This model is then to be used in the templates as `m.comment`. Be careful to give your models a unique name to prevent name clashes with other models and Erlang modules.

See also:

Models (page 23)

templates/

This directory contains all *Templates* (page 19). Templates do not have any prefix in their name, as they are not (directly) compiled as Erlang modules.

The following naming conventions for templates are used:

- All templates have the extension ".tpl"
- Templates used as a complete page can have any name: "my_special_page.tpl"
- Templates used as the base of other templates, using the *extends* (page 445) tag, have the word "base" in them: "base.tpl"; "email_base.tpl".
- Templates only used by including them in other templates start their name with an underscore: "_example.tpl"
- The template for the home page of a site is called "home.tpl"
- Templates for displaying resources are called "page.tpl"

See also:

Templates (page 19)

filters/

This directory holds Erlang modules, each of which defines a *template filter* (page 23).

Each filter must have an unique name, reflecting the filter's name. For example, the filter "tail" resides in the Erlang module `filter_tail.erl` and exports the function `tail/1`. Filters are added in the filters directory. The template compiler will insert references to the correct modules into the compiled templates. A missing filter will result in a crash of the compiled template.

See also:

Filters (page 23)

validators/

This directory holds Erlang modules, each of which defines a *validator* (page 27).

Validators are prefixed in the same way as actions and scomps, except that the word "validator" is used. For example the validator "email" in the module "mod_base" has the file name: "validator_base_email.erl"

See also:

Forms and validation (page 27)

services/

The services folder holds Erlang modules, each of which functions as an API method that you can use to access Zotonic from another application. These are invoked by *controller_api* (page 349).

Services are named a bit differently: the name of the module is *always* used in the service name: The service `base/export` will be found in the file `mod_base/services/service_base_export.erl`. This particular service can then be found at `http://yoursite.com/api/base/export`.

See also:

API services (page 51) and *controller_api* (page 349)

Changing / recompiling files

Changes to the Erlang files in a module are visible after issuing the `zotonic update` CLI command, or `z:m().` from the Zotonic shell. Any new lib or template files, or changes in the dispatch rules are visible after the module indexer has rescanned all modules. You can do this with the "rescan modules" button on the modules page in the admin. Changes to templates are directly visible.

Priority

A site's mod_prio metadata attribute is usually set to 1, to make sure that it is the first module where Zotonic looks for *template lookups* (page 19) and the like.

Dependencies

Modules can have dependencies on other modules. These are expressed via the module's metadata, as follows:

```
-mod_depends([mod_admin]).
```

This states that the current module is dependent on `mod_admin` to be installed.

Sometimes, explicitly depending on a module name is not a good idea: there might be more modules that perform the same functions but are competing in implementation. In that case, such modules can export a `mod_provides` meta tag, so that dependent modules can *depend* on what one of these modules *provide*.

Example: `mod_a` and `mod_b` both provide some functionality, `foo`:

```
-module(mod_a).
-mod_provides([foo]).
```

and:

```
-module(mod_b).
-mod_provides([foo]).
```

Now, another module, `mod_bar`, needs the "foo" functionality:

```
-module(mod_bar).
-mod_depends([foo]).
```

Now, the module manager will require either (or both!) of the `mod_a` and `mod_b` modules to be activated, before `mod_bar` can be activated.

A module automatically *provides* its own module name, as well as its name minus the `mod_`. So, `mod_bar` has implicitly the following *provides* constructs:

```
-module(mod_bar).
-mod_provides([mod_bar, bar]).
```

These two provides are there even when a module adds its own `provides` clauses.

Module startup order

Note that when a site start, its modules are started up in order of module dependency, in such a way that a module's dependencies are always started before the module itsef starts.

Module versioning

Modules can export a `-module_schema()` attribute which contains an integer number, denoting the current module's version. On module initialization, `Module:manage_schema/2` is called which handles installation and upgrade of data.

Minimal example:

```
-module(mod_twitter).
-mod_title("Twitter module").
-mod_schema(3).  %% we are currently at revision 3

-export([manage_schema/2]).
.... more code here...

manage_schema(install, Context) ->
% .. code to install your stuff here, for instance:
#datamodel{categories=
              [
               {tweet,
                text,
                [{title, <<"Tweet">>}]}
              ]};

manage_schema({upgrade, 2}, Context) ->
%% code to upgrade from 1 to 2
ok;

manage_schema({upgrade, 3}, Context) ->
%% code to upgrade from 2 to 3
ok.
```

Note that the install function should always be kept up-to-date according to the latest schema version. When you install a module for the first time, no upgrade functions are called, but only the `install` clause. The upgrade functions exist for migrating old data, not for newly installing a module.

Data model notification

In the `#datamodel` record you can manage categories, predicates, resources, media and edges. You can also set the `data` property, which will send out a first *notification* (page 46). To subscribe to that notification, export `observe_manage_data/2` in your site or module.

Using categories defined by other modules

When your site needs to add resources which are defined by other module's `manage_schema` functions, you need to make sure that those modules manage functions are called first. This can be realised by adding a dependency to those modules, as explained in *Module startup order* (page 43).

For instance, when you want to create a custom menu for your site:

```
manage_schema(install, _Context) ->
    #datamodel{
        resources=[
            {help_menu, menu, [
                {title, "Help"},
                {menu, [...]}
            ]}
        ]
    }.
```

You also need to make sure that you add a *dependency* (page 43) to `mod_menu`, which creates the `menu` category for you:

```
-mod_depends([mod_menu]).
```

gen_server based modules

When you need a running process, e.g., a module that does something in the background, then it is possible to implement your module as a gen_server. A gen_server is a standard way to implement a reliable Erlang worker process.

In that case you will need to add the behaviour and gen_server functions. You also need to change the `init/1` function to accept an property list, which contains the site definition and a `{context, Context}` property.

This server module will be started for every site in a Zotonic system where the module is enabled, so it can't be a named server.

See also:

Erlang gen_server principles

A minimal example

```
-module(mod_example).
-author("Nomen Nescio <nomen@example.com>").

-behaviour(gen_server).

-mod_title("Your module title").
-mod_description("Description what this module does.").
-mod_prio(500).

-export([init/1, handle_call/3, handle_cast/2, handle_info/2, terminate/2, code_
↪change/3]).
-export([start_link/1]).

-include_lib("zotonic.hrl").
-record(state, {context}).


%% Module API

start_link(Args) when is_list(Args) ->
    gen_server:start_link(?MODULE, Args, []).

%% gen_server callbacks

init(Args) ->
    {context, Context} = proplists:lookup(context, Args),
    {ok, #state{context=z_context:new(Context)}}.
```

```
handle_call(Message, _From, State) ->
    {stop, {unknown_call, Message}, State}.

handle_cast(Message, State) ->
    {stop, {unknown_cast, Message}, State}.

handle_info(_Info, State) ->
    {noreply, State}.

terminate(_Reason, _State) ->
    ok.

code_change(_OldVsn, State, _Extra) ->
    {ok, State}.
```

As you can see, this code is almost identical to the standard Erlang `gen_server` boilerplate, with the exception of the metadata on top.

You also see that the `start_link/1` function is already implemented. Note that in this function the gen_server is started without registering the server under a name: this is done because the module can be started multiple times; once for each site that needs it.

The `init/1` function contains some more boilerplate for getting the `context{}` argument from the arguments, and storing this context into the server's state. This way, you'll always have access to the current context of the site in the rest of the gen_server's functions.

Notifications

Zotonic's notifier system makes it possible to create modular components with a pluggable interface. The notifier system is used by internal core Zotonic components like the authentication mechanism, the logging system and more.

The notification system can not only act as a traditional event subscription system but also as an advanced priority based function dispatch system. It uses the same priority system which is used to select templates. This makes it possible to override pre-defined default behaviour of core Zotonic modules.

A notification message is a tagged tuple. The first element of the tuple is the type of notification message. An observer can use this to subscribe to specific messages it is interested in.

Example:

```
{acl_logon, 1234}
```

Or:

```
{module_activate, mod_stream, <0.32.0>}
```

You can find a *list of all notifications* (page 486) in the reference section.

Sending notifications

As mentioned earlier, the notification system can not only be used to just send events to observers. Observers can also return values back. They can do this in various ways described in the methods below.

notify Send a message to all observers. This is used if you want to notify other observers about a specific event. In Zotonic this is used a lot. For instance, it is used to notify modules of about user logons, or notify when modules are activated and deactivated.

notify1 Notify the first observer. This is useful for if you want to be sure just one observer can do something with the message.

first Call all observers, and use the first non `undefined` answer. This is used to get information from one of the observers. By using the notification system it makes sure that modules are decoupled.

map Call all observers and get a list of all answers.

foldl Do a fold over all observers, high prio observers first.

foldr Do a fold over all observers, low prio observers first.

You can also send notifications from JavaScript.

Subscribing to notifications

Registering as an observer works as follows:

```
z_notifier:observe(NotificationType, Handler, Priority, Context)
```

If the module is either a Zotonic module or a site module, the `Priority` parameter can be omitted. The observer will then get the same priority as the module.

NotificationType The type of notification you want to observe, an atom.

Handler Can be a `pid()`, or a `{Module, Fun}` tuple. When the handler is a `pid()` and the notification is sent with `notify` or `notify1` the gen_server process receives a `handle_cast`. When an answer is expected back `handle_call` is used. This is the case for `first`, `map`, `foldl` and `foldr`.

Priority The priority of the observer. This influences the order in which they are called.

Context The Zotonic context.

Example:

```
z_notifier:observe(acl_logon, {mysitewww, handle_logon}, Context)
```

Subscription shorthands

Modules and sites can use shortcuts for registering as an observer. When the Zotonic module exports a function with the prefix `observe_` or `pid_observe_` Zotonic's module manager will register the observer for you.

For example exporting `observe_acl_logon/2` will register that function as an observer. It will be triggered when the `acl_logon` notification is fired.

Handling notifications

When a notifications is sent the `z_notifier` module looks in its tables to see if there are any observers interested in receiving it. There are three types of notifications.

Cast notifications This is the simplest notification. The notifier does not expect an answer back the result of the handler is ignored. This kind of notification is triggered by calling `z_notifier:notify/2` or `z_notifier:notify1/2`. They are useful for letting other modules know about a certain even or condition. This makes it possible for other modules to act on it.

For example, *mod_development* (page 289) uses call notifications to trigger builds and reloads. By doing this other modules can notify `mod_development` to trigger builds. But when `mod_development` is disabled nothing will happen.

Call notification For this kind of notification, `z_notifier` expects an answer back. This answer is returned back to the notifier. This kind of notifications is used to decouple modules. For instance a module can ask another module for a special URL to go to after logging in without knowing which module will do this. Call notifications are triggered by: `z_notifier:first/2` and `z_notifier:map/2`.

For example, *mod_signup* (page 311) uses a call notification to find out what page to redirect to after a successfull signup. This allows one to customize the signup process.

Fold notifications

Fold notifications are called, with `z_notifier:foldl/3` or `z_notifier:foldr/3`. It works similarly to the lists:foldr and lists:foldl functions of Erlang's lists module.

The fold function calls each observer in sequence, either starting at highest (`foldl`) or at lowest (`foldr`) priority, passing values and an initial accumulator value.

Each observer can adapt values in the accumulator, and needs to return it, for passing on to the next observer. The final value of the accumulator is returned as result. This is useful if you want multiple modules to be able to adapt and use values in the accumulator.

For example, *mod_admin* (page 275) uses a fold notification (called `admin_menu`) to build up the admin navigation menu, where each observer is called to add menu entries to the menu.

See also:

list of all notifications (page 486)

Browser/server interaction

There are multiple ways to set up interaction between server-side Zotonic code and client-side JavaScript.

If you want to initiate the interaction in the client (browser)

1. wired events
2. notifications
3. API methods
4. transport
5. publish/subscribe.

Wired events

First, define a named action:

```
{% wire name="test" action={growl text="Hello world"} %}
```

Then, call that action from your JavaScript code:

```
z_event("test");
```

Trigger notifications from JavaScript

Trigger a *notification* (page 46) in Zotonic with the `z_notify` JavaScript function:

```
z_notify("mymessage");
```

or:

```
z_notify("mymessage", {foo: bar, who: "world"});
```

This will trigger a call to:

```
z_notifier:first(#postback_notify{message="mymessage"}, Context)
```

Which you can handle in any Zotonic module by defining:

```
-export([observe_postback_notify/2]).
observe_postback_notify(#postback_notify{message="mymessage"}, Context) ->
    Who = z_context:get_q(who, Context),
    z_render:growl("Hello " ++ Who, Context);
observe_postback_notify(_, _Context) ->
    undefined.
```

All arguments are available via the `z_context:get_q/2` function (and friends).

Calling API services

A third way is to write your own *API services* (page 51) and use standard JavaScript to perform Ajax GET/POST requests from the browser.

This use is perfectly possible and legal, although the other methods are preferred, as they integrate nicely with the notification and action systems. The API is more targeted to other applications interfacing to Zotonic.

Transport

The transport functions are a low-level layer, just above the WebSocket, Comet and AJAX methods used for communication between the server and the browser.

Zotonic has a message bus to transport data between server and browser. It transports structured data in different formats and supports retransmission in case of lost messages.

From browser to server

To send a message from the browser to the server:

```
z_transport("mod_example", "ubf", {hello: "world"});
```

And then on the server, use Erlang to process the message:

```
-module(mod_example).

-export([event/2]).

-include_lib("zotonic.hrl").

event(#z_msg_v1{data=Data}, Context) ->
    io:format("~p", [Data]),
    Context;
```

This will print on the console:

```
[{<<"hello">>,<<"world">>}]
```

Quality of service

The message will be sent with a quality of service of 0. That means the browser will try to send the message, but will not check if it arrived. Alterntively, you can send with a qos of 1, in that case the browser will wait for an ack, and if that doesn't arrive in 30 seconds, then a duplicate message will be requeued for transport:

```
z_transport("mod_example", "ubf", {hello: "world"}, {qos: 1});
```

It is possible to define a callback function that will be called if an ack is received:

```
z_transport("mod_example", "ubf", {hello:"world"}, {
    qos: 1,
    ack: function(ackMsg, callOptions) {
        alert(ackMsg);
    }
});
```

From server to browser

Sending JavaScript (or other data) from the server to the browser is straightforward:

```
z_transport:page(javascript, <<"alert('Hello World');">>, Context);
```

This transports the JavaScript to the page associated with `Context`. This JavaScript will then be evaluated in the browser.

The default quality of service is 0 (see above); to let the page queue retry delivering the message it is possible to specify another quality of service:

```
z_transport:page(javascript, <<"alert('Hello World');">>, [{qos, 1}], Context);
```

It is also possible to send a message to all open pages of a session, or to all sessions of a user:

```
z_transport:session(javascript, <<"alert('Hello World');">>, [{qos, 1}], Context);
z_transport:user(javascript, <<"alert('Hello World');">>, [{qos, 1}], Context);
```

Or transport to a specific page, session or user, but then you will need to specify the message and the message-queue:

```
Msg = z_transport:msg(session, javascript, <<"alert('Hello World');">>, [{qos, 1}
↪]).
z_transport:transport_user(Msg, UserId, Context).
```

The message queue is either `session` or `page`. It defines which queue will be responsible for resending the message and where the ack message is received. If `user` is specified as queue then it will be replaced by `session`.

See also:

transport reference (page 498).

Publish/subscribe (PubSub)

It is possible to publish and subscribe to topics on the server. Messages are relayed between the server and the browser.

See *mod_mqtt* (page 301) for more information.

An example of MQTT PubSub usage is the custom tag scomp-live.

Transport mechanisms

Zotonic uses various mechanisms to transport data between the browser and the server:

- AJAX callbacks to the server using *controller_postback* (page 359)
- WebSocket with bidirectional transports using *controller_websocket* (page 365)
- Comet transporting data from the server to the browser using *controller_comet* (page 350)
- HTML form posts to *controller_postback* (page 359).

AJAX calls also transport back data from the server to the browser.

Publish/subscribe

Publishing and subscribing to topics is done through MQTT.

API services

Services provide a generalized way to create API calls. These calls automatically use the authentication mechanism (session id or *OAuth* (page 306)) to perform access checks.

A Zotonic service is created in a separate service (sub)module. Here you typically define the method name, the type of access (GET or POST) and if authentication is required.

How service names are mapped to URLs

The URL to call an API service is defined by the module and the method name.

In a vanilla Zotonic install, there is one single URL namespace under which all API services can be accessed. *controller_api* (page 349) by default intercepts all URLs according to the following patterns:

```
/api/:module/:method
/api/:module
```

On these URLs, a lookup is done to find the corresponding Zotonic module inside the services subdirectory of the module:

```
mod_modulename/
    mod_modulename.erl
        services/
            service_modulename_methodname.erl
```

If no method name is used in the `/api/:module` URL, the method name will be equal to the module name - see `/api/search` in the table below.

Examples of URLs that correspond to service handlers:

URL	Module	Method	Located in service .erl file
/api/base/export	mod_base	export	mod_base/services/service_base_export.erl
/api/base/info	mod_base	info	mod_base/services/service_base_info.erl
/api/search	mod_search	search	mod_search/services/service_search_search.erl

For creating services at alternative URLs, see *Creating services at a non-standard URL* (page 349) in the *controller_api* (page 349) documentation.

Service naming in detail

As stated above, a service module is defined as:

```
service_<modulename>_<processname>.erl
```

And is then reachable on the URL `http://<hostname>/api/<module_name>/<process_name>`.

The module *module_name* to be activated for the API call to work.

If you have a module named *mod_something* that needs a service to return stats, your directory would look like this:

```
mod_something/
    mod_something.erl
        services/
            service_something_stats.erl
```

The url for this service will be `http://<site_addr>/api/something/stats`

The key is that an activated module (minus the `mod_` prefix if you use them!) should be part of the service name. Zotonic parses the service modules filename to identify what module a service relates to and what process should be called. It checks to make sure that module is activated and it also uses that same information when matching a service url. So, reversly, `service_something_stats.erl` is served by `http://<hostname>/api/something/stats`.

Service metadata

Like stated, any service is a regular *Erlang module*. There are however a few extra attributes for use in the service which describe it more. Firstly, there is `svc_title`:

```
-svc_title("Retrieve uptime statistics.").
```

The title of a service should be a human-readable, one-line description of what the services does. This title is used in the OAuth authentication dialog: when authorizing an application, the titles of the services that it wants to access are listed, for the authorizing user's consideration.

Secondary there is `svc_needauth`:

```
-svc_needauth(false).
```

This is a boolean value which tells the system whether or not a user needs to be authorized in order to use the service.

If authentication is needed for a service, a service can only be accessed either by using the session cookie or by using an authorized OAuth (1.0a) token.

Creating a GET service

By implementing the `process_get/2` function in your service module, it indicates that it is able to handle GET requests. A full example of a services which handles a GET request is listed below:

```
-module(service_something_stats).
-author("Arjan Scherpenisse <arjan@scherpenisse.net>").

-svc_title("Retrieve uptime statistics of the system.").
-svc_needauth(true).

-export([process_get/2]).

-include_lib("zotonic.hrl").

process_get(_ReqData, _Context) ->
    Stats = [{count, 12310}, {uptime, 399}],
    z_convert:to_json(Stats).
```

This module could be called `service_something_stats.erl` and then gets served at `/api/something/stats`. Its output is a JSON object containing a `count` and an `uptime` field, containing some values.

Of course, you would write real code there which retrieves actual stats. If your module `something` contains the function `stats_data/1`, call it from the process function like this:

```
process_get(_ReqData, Context) ->
    Stats = mod_something:stats_data(Context),
    z_convert:to_json(Stats).
```

Creating a POST service

Similar to GET, by implementing the `process_post/2` function in your service module, it indicates that it is able to handle POST requests. The POST parameters are accessible to you by using `z_context:get_q/2`.

A full example of a services which handles a POST request is listed below:

```
-module(service_something_process).
-author("Arjan Scherpenisse <arjan@scherpenisse.net>").

-svc_title("Processes the given id.").
-svc_needauth(true).

-export([process_post/2]).

-include_lib("zotonic.hrl").

process_post(_ReqData, Context) ->
    Id = z_context:get_q("id", Context),
    %% Do some processing here...
    Response = [{result, Id}],
    z_convert:to_json(Response).
```

This module could be called `service_something_process.erl` and then gets served at `/api/something/process`. It requires authentication, and is only accessible with POST and expects an `id` argument to be posted.

Again, its output is a JSON object containing a `result` field.

Setting response headers

You can set response headers by returning a `{Result, #context{}}` tuple from the `process_get/2` and `process_post/2` calls:

```
process_get(_ReqData, Context) ->
    Stats = mod_something:stats_data(Context),
    Result = {struct, [{count, 100}]},
    Context1 = z_context:set_resp_header("Cache-Control", "max-age=3600", Context),
    {Result, Context1}.
```

Uploading files

The simplest way to upload files is to use the ready-made API service service-media_upload. But if you want to have different behavior (for instance to connect an uploaded user picture to a user page), it is easy to create your own.

The post payload should be `multipart/form-data` encoded (which is the standard for file uploads).

The posted data is automatically retrieved by Zotonic and made available via `z_context`. If you use `"upload"` for the form data `name` field, you get the upload data from `z_context:get_q("upload", Context)`. The resulting value is an `#upload{}` record, and can be passed directly to `m_media:insert_file`:

```
Upload = z_context:get_q("upload", Context),
m_media:insert_file(Upload, Context)
```

Error handling

An HTTP status error code will be generated when `process_get` or `process_post` returns an error object:

```
{error, error_name, DetailsString}
{error, error_name, DetailsString, ErrorData}
```

Additionally, you may also `throw()` these error structures inside `process_get` and `process_post`, to easily short-circuit your error handling (e.g. for input validation):

```
Title = z_context:get_q("title", Context),
z_utils:is_empty(Title) andalso
    throw({error, missing_arg, "title"}),
```

Simple error feedback

By providing the error name, a corresponding HTTP status code and message will be set. Supported error names are:

Name	Generated message	Status code
`missing_arg`	Missing argument: + Details	400
`unknown_arg`	Unknown argument: + Details	400
`syntax`	Syntax error: + Details	400
`unauthorized`	Unauthorized.	401
`access_denied`	Access denied.	403
`not_exists`	Resource does not exist: + Details	404
`unprocessable`	Unprocessable entity: + Details	422
(other)	Generic error.	500

For example:

```
process_post(_ReqData, Context) ->
    %% Do some processing here...
    case Error of
        true ->
            {{error, missing_arg, "username"}, Context};
        false ->
            {z_convert:to_json(Data), Context}
    end.
```

Working with Error Objects

In some cases it is useful to return more detailed error feedback. The JSON API has specified a format for this. The thinking behind the format is that the server, after encountering an error, may continue to process information, and instead of returning a single error code, returns multiple found errors.

Taking this approach, this error information is returned as a JSON array, with a top key entry `errors`:

```
["errors": {
    "detail": "...",
    "source": "...",
    "status": "...",
    "title": "..."
}]
```

Of course there is no obligation to use JSON API structure, but if you want, the code of one of those functions - for instance to log on - could look like this:

```
case User of
    undefined ->
        {error, [
            {status, 422},
            {source, "mod_webapp:logon"},
```

```
            {title, "No user found"},
            {detail, "Could not log on user"}
        ]};
    _ ->
        {ok, User}
end.
```

The return data of multiple functions may then be aggregated into a single error data object and returned as a list of Error Objects:

```
process_post(_ReqData, Context) ->
    %% Do some processing here...
    %% Accumulate all data...
    %% Handle return:
    case Data of
        {error, ErrData} ->
            {{error, unprocessable, "", z_convert:to_json(ErrData)}, Context};
        _ ->
            {z_convert:to_json(Data), Context}
    end.
```

Enabling Cross-Origin Resource Sharing (CORS)

By default the server has a same-origin policy: scripts that access the API must reside on the same server.

Cross-origin resource sharing allows cross-domain requests for apps outside of the server domain. CORS header settings define which requests are (and are not) allowed.

In-depth background information is available at https://developer.mozilla.org/en-US/docs/Web/HTTP/Access_control_CORS

CORS settings are defined in the site's config.

Site config settings

`{service_api_cors, false}` Set to `true` to enable CORS

`{'Access-Control-Allow-Origin', "*"}`

`{'Access-Control-Allow-Credentials', undefined}`

`{'Access-Control-Max-Age', undefined}`

`{'Access-Control-Allow-Methods', undefined}`

`{'Access-Control-Allow-Headers', undefined}`

Note:

- The config file can be modified without a site restart.
- The "Access-Control" settings only work if `service_api_cors` is set to true.
- The setting name is an Erlang atom and must be in single quotes.
- Setting values are either `undefined` or a string value. Multiple values can be set as a comma-separated string, for instance:

```
{'Access-Control-Allow-Headers', "authorization, X-Requested-With, Content-Type
↪"}
```

Service authentication

Like stated, authentication and authorization is done either through the Zotonic session or through a custom notification hook, `#service_authorize{}`.

For session authentication, you need to have a valid session id (`z_sid`) cookie. This method of authentication is the easiest when you are accessing the services from JavaScript from the same domain as your user is logged in to.

When no session is available, but the called services requires authentication (according to its `svc_needauth` metadata attribute), a *notification hook* (page 46) with the name `service_authorize` is called.

In a default Zotonic install, this `service_authorize` hook is handled by the *OAuth module* (page 306), but can be replaced by a different service authentication module.

The module implementing the `service_authorize` hook is expected to return either *undefined* (when the request is not applicable) or a response which must conform to the Webmachine `is_authorized/2` return format.

See also:

- *Services glossary entry*
- *List of all core services* (page 435)
- *mod_oauth* (page 306)
- *controller_api* (page 349)

E-mail handling

Configuration

Any Zotonic system is capable of sending and receiving e-mail messages over SMTP.

Zotonic implements a mailing system for sending text and HTML messages to one or more recipients.

Out of the box, e-mail sending should "just work".

Feedback

If you need feedback on messages that have been sent, enable *mod_logging* (page 298) which provides an overview of sent/received and bounced messages.

Site-specific settings

Module	Key	Value
site	email_from	Set this to the from-address you want to e-mail to appear from, e.g. something like noreply@yoursite.com.
site	email_override	If set, all e-mail messages that get sent from Zotonic will arrive at this address. Useful if you are testing but don't want to confuse other people with your test e-mails.
site	smtphost	The hostname where you want messages to appear from. Mostly used for bounce message handling and the EHLO handshake. Defaults to the site's hostname, but can be overriden
site	admin_email	E-mail address of the admin user, the address where admin log/debug messages get sent to when using `z_email:send_admin/3`.
site	bounce_email_override	E-mail address where bounces are sent to. Normally a special bounce address is generated. See also the discussion about `smtp_bounce_email_override` below.

The `z_email:send_admin/3` command actually looks in three different places for determining the admin e-mail address: the config key `zotonic.admin_email`, then the `site.admin_email` key, and finally the *email* property of the admin user (user with id 1).

If no admin email address is found then the address `wwwadmin@example.com` is used, where `example.com` will be your site's hostname.

Zotonic-wide settings

The file `~/.zotonic/zotonic.config` can be configured to hold any of the configuration options below. They are in effect for every site running in the Zotonic instance.

Zotonic-wide settings for sending email

Key	Description
smtp_relay	Whether or not to use a SMTP relay host. Boolean value, defaults to false.
smtp_host	The hostname for the SMTP relay host, only needed if smtp_relay is enabled.
smtp_ssl	Whether or not to use SSL on the relay host, only needed if smtp_relay is enabled.
smtp_username	The username for the relay host, only needed if smtp_relay is enabled.
smtp_password	The password for the relay host, only needed if smtp_relay is enabled.
smtp_no_mx_lookups	Set to true to not do a MX lookup before sending mail. (default: false)
smtp_verp_as_from	Use the "from" address as VERP for bounce handling (default: false)
smtp_bcc	Optionally send a BCC of every sent to this address
email_override	A global e-mail override. The override logic first checks the site override, and then the global override address. Useful for testing and development.
smtp_bounce_domain	Which domain to use for bounce VERP messages. Defaults to the smtp domain of the site sending the email.
smtp_bounce_email_override	The e-mail address for bounce handling. Only use if all else fails (see the paragraphs after the next one).

Zotonic-wide settings for receiving email

To receive email the SMTP server has to listen on the correct IP address and port. Spam filtering is done by checking DNSBL (DNS Block List) servers and optionally using Spamassassin.

Key	Description
smtp_listen_domain	The domain announced in the HELO
smtp_listen_ip	IP address to listen on for incoming SMTP connections. Defaults to `"127.0.0.1"`
smtp_listen_port	IP address to listen on for incoming SMTP connections. Defaults to 2525.
smtp_dnsbl	List for the DNS block lists used for checking incoming email connections. Defaults to `["zen.spamhaus.org","dnsbl.sorbs.net"]`
smtp_spamd_ip	Optional IP address for a spamassassin host
smtp_spamd_port	Optional port number for a spamassassin host

The sender's domain

Recipients of e-mail want a valid sender's address on the envelope. This section describes how to set your e-mail bounce/sender domain and fixing domain errors when sending e-mail.

You have to think of e-mail as normal snail mail. There is a message and an envelope.

The `email_from` is the address that is written on the *message*. It could be anything, and is generally not used when delivering your mail. Just like with snail mail, the postman only looks on the *envelope*, not on the message.

The address on the envelope is the most important address. It is where your e-mail is returned to when the message can't be delivered (so called bounces). Often it is also checked for validity when the e-mail is delivered at a SMTP server.

You need a valid domain for this envelope sender address. The part before the `@` is generated by Zotonic and is used for identifying the original message and recipient when the message bounces.

If the generated part is not acceptable then you can force an envelope address by setting the `smtp_bounce_email_override` option. Setting the bounce/envelop address manually disables Zotonic's build-in handling of bounces that happen *after* the e-mail was accepted for delivery by the remote SMTP host.

The bounce e-mail address can also be set on a per-site basis using the `site.bounce_email_override` configuration. See the site specific settings table above.

How does Zotonic know the domain?

It checks in order:

- site's config: `bounce_email_override` (you can also set this with the admin config as site.bounce_email_override)
- global `~/.zotonic/zotonic.config`: `smtp_bounce_domain` setting
- site's config: `smtphost`
- site's config: `hostname`

Any *bounce_email_override* configuration must be a complete email address. For example: `bounces@example.org`

If no *bounce_email_override* is used then the part before the `@` is generated by Zotonic itself, for administration and detection of bounces. A typical sender address on the envelope looks like: `noreply+mlcm6godbz2cchtgdvom@example.org`

Sending E-mail

Once configured, you can use the following Erlang commands to send e-mail from Zotonic code:

Command	Explanation
`z_email:send_admin/3`	Sends a quick e-mail to the site administrator. Handy to notice the site admin that something is wrong, a job has finished, etc... The e-mail that is used is the admin_email address that is specified in the site's config file.
`z_email:send/4`	Sends a text message with a subject to a specified recipient.
`z_email:send_render/4`	Renders a template and sends it as a HTML message to a specified recipient.
`z_email:send/2`	Sends an email defined by a `#email{}` record.

Receiving E-mail

In its default configuration, Zotonic starts an SMTP server on port 2525 for receiving e-mail messages. You can write your own code to decide what happens if somebody sends e-mail to the system, by implementing the `email_received` notification (see below).

The SMTP server is also used to receive bounce messages from other servers, when sending of a message has failed. *mod_mailinglist* (page 299) uses this functionality to automatically deactivate invalid e-mail addresses.

Configuring incoming E-mail

To send messages to Zotonic, the domain part of the e-mail address should have an A or MX record which points to the server where Zotonic is able to receive on port 25. This means that you have to add a firewall rule to redirect port 25 to 2525.

If you were to set up e-mail receiving for a site called `example.com`, you could test if this is working by using the *netcat* program, like this:

```
nc example.com 25
```

Then, you should be greeted by Zotonic in the following way:

```
220 example.com ESMTP Zotonic 0.13.0
```

Press ctrl-c to exit.

Handling incoming E-mail

When receiving an e-mail message, Zotonic looks at the domain part of the e-mail address to determine which *Zotonic site* is configured to handle this message. It looks at the `host` and `hostalias` fields in the site's config file to match the recipient domain.

If no site matches the e-mails domain, the message is dropped, and a warning logged.

For handling incoming messages in your site, you need a hook in your site module to do something with the received messages, implementing the `email_receive` notification.

The code in your module looks like this:

```
observe_email_received(E, _C) ->
    lager:warning("Email from: ~p: ~p", [E#email_received.from,
                                         E#email_received.email#email.subject]),
    ok.
```

Export this function and then restart your site. Now, send an e-mail message to any address `@example.com`, and notice that it arrives in Zotonic:

```
(zotonic001@host.local)9> 20:57:54.174 [warning] Email from: <<
↪"arjan@miraclethings.nl">>: <<"Hello!">>
```

Troubleshooting

Check in the admin the log and smtp log. If a message bounces back to the Zotonic SMTP server, you will see errors there. A typical error looks like this:

```
SMTP: bounce: 504 5.5.2 <noreply+mlcm6godbz2cchtgdvom@oeps>: Sender address␣
↪rejected: need fully-qualified address To: piet@example.com (1234) From:
↪<noreply+mlcm6godbz2cchtgdvom@oeps>
```

Command-line shell

The Zotonic shell gives you access to a running Zotonic instance with its code and data.

To connect to the background Zotonic server instance within an EShell (Erlang shell):

```
zotonic shell
```

Alternatively, use the *debug* commmand to launch the Zotonic server interactively and get an EShell on the running instance:

```
zotonic debug
```

The `start.sh` command in the root folder is a shortcut for this command.

See also:

Command-line (page 496)

Tab completion

The Zotonic shell has tab completion.

z_<tab> Lists Zotonic library modules.

m_<tab> Lists Zotonic models.

mod_<tab> Lists Zotonic modules.

Add a colon to get all available functions from a module, for instance: `m_rsc:<tab>`.

Often used commands

The `z` module provides shortcuts for the most used commands. From within the Zotonic shell:

z:m(). (Re)makes all erlang source modules and resets the caches.

z:flush(sitename). **or** **z:flush().** Resets all caches, reloads the dispatch rules and rescans all modules.

z:ld(modulename). **or** **z:ld().** Reloads an Erlang module, or reloads all changed Erlang modules

z:restart(sitename). **or** **z:restart().** Restarts the site, or restarts Zotonic

z:c(sitename). Gets the current site context. This call is often used inside another function call - see examples below.

Activating/deactivating modules

z_module_manager:deactivate(mod_modulename, z:c(sitename)). Deactivates a module.

z_module_manager:activate(mod_modulename, z:c(sitename)). Activates a module.

Resetting a user password

Sometimes it happens that you want to reset an user's password from the Erlang shell. You can do this from the Erlang shell without using the admin or the reset-password mail/dialog.

m_identity:set_username_pw(1234, "username", "password", z:c(sitename)).
Where 1234 is the rsc id of your user (1 for the admin), this must be an integer.

Accessing data

Sometimes it is desirable to access data stored in Zotonic from the EShell. This is useful for experimenting with code without having to go through an involved process or through HTTP requests to test it.

These calls use the site context as parameter. Get it using:

```
Context = z:c(sitename).
```

The `m_rsc` model module provides functions for retrieving and interacting with pages and other resources stored in Zotonic's datastore for the site.

`RscProps = m_rsc:get(Id, Context).` Returns the entire resource as a proplists structure. Id is the resource Id number.

```
Title = m_rsc:p(Id, title, Context).
```

You can also use the Erlang proplists module:

```
Title = proplists:get_value(title, RscProps).
```

Searching

To perform a search on the running site, use the `z_search` module.

`rr(z_search).` Loads all records defined in `z_search` (including those defined in include files such as the `#search_result` record).

`Results = z_search:search({latest, [{cat, text}]}, Context).` Returns the search result record, for the search on pages from the category *text*. You can specify the category as a string, binary, atom or integer.

To retrieve the list of pages, we access the `result` property of the record data:

```
Pages = Results#search_result.result.
```

Use `m_rsc:get(Id, Context)` to retrieve Page information of each search result (see above).

Debugging

We have added the `recon` application to the zotonic deps. This allows one to easily debug many aspects of a running zotonic node. It contains tools for tracing calls, check memory and bandwidth usage and a lot more. For more information see: Stuff Goes Bad: Erlang in Anger.

Testing sites

It is possible to create end-to-end integration tests for Zotonic websites. Tests like these are called *sitetests*. They run within the Zotonic shell, starting the website under test using a special database schema so that the main database is not affected by the tests.

To run the sitetests for a site called `example`, run the following command:

```
z_sitetest:run(example).
```

This will stop the `example` site, create a new database schema (called `z_sitetest`), start the site (which installs all the site's data, e.g. those defined by the *Module versioning* (page 44), into the new schema), and then scans all compiled Erlang modules for modules named `example_*_sitetest.erl`. These found test modules are then run using Erlang's standard EUnit test framework.

Note: The filename pattern of the tests is *(sitename)_(testname)_sitetest.erl*, where *sitename* is the name of the site under test (lowercase alphanumerics + underscores), and *testname* is a name for the test suite (lowercase alphanumerics *only*). *testname* **cannot** contain any underscores.

Example `sitetest` module

Put the following inside the `example` site under the filename `tests/example_administrator_sitetest.erl`:

```
-module(example_administrator_sitetest).
-compile(export_all).

-include_lib("eunit/include/eunit.hrl").

sudo_context() ->
  z_acl:sudo(z:c(example)).

administrator_name_test() ->
  ?assertEqual(<<"Site Administrator">>, m_rsc:p(1, title, sudo_context())),
  ok.
```

Test Driven Development

Running the sitetests is integrated into the filewatcher hooks. As soon as you edit a `*_sitetest.erl` file, all tests in that specific sitetest file will be executed.

While developing a site it is often handy to continuously run the tests while you are developing the site.

To establish this, you can call `z_sitetest:watch(site)` to start watching all Erlang files inside the site under development. When the filetest is watching a site, all filetests are triggered as soon as any Erlang module inside your site is being recompiled.

To stop the automatic test running again, call `z_sitetest:unwatch(site).`

Example testing output

Running the test command `z_sitetest:run(example).`, will produce output similar to the following:

```
(zotonic001@host)33> z_sitetest:run(example).
15:45:18.162 [info] Site stopped: example (<0.17763.0>)
15:45:18.682 [warning] [example] Database connection failure: noschema
15:45:18.688 [warning] [example] Creating schema "z_sitetest" in database "example"
15:45:18.693 [info] [example] Retrying install check after db creation.
15:45:18.702 [warning] [example] Installing database with db options: [{dbschema,
↪"z_sitetest"},{dbdatabase,"example"},{dbhost,"localhost"},{dbport,5432},{dbuser,
↪"zotonic"}]
15:45:19.226 [info] example: Install start.
15:45:19.226 [info] Inserting categories
15:45:19.257 [info] Inserting base resources (admin, etc.)
15:45:19.264 [info] Inserting username for the admin
15:45:19.267 [info] Inserting predicates
15:45:19.290 [info] example: Install done.
15:45:19.299 [info] Site started: example (<0.22082.0>)
15:45:19.575 [info] [example] info @ z_datamodel:169  Creating new category 'menu'
15:45:19.604 [info] [example] info @ z_datamodel:169  Creating new menu 'main_menu'
15:45:19.723 [info] [example] info @ z_datamodel:169  Creating new category
↪'content_group'
15:45:19.755 [info] [example] info @ z_datamodel:169  Creating new content_group
↪'system_content_group'
15:45:19.775 [info] [example] info @ z_datamodel:169  Creating new content_group
↪'default_content_group'
15:45:19.893 [info] [example] info @ z_datamodel:169  Creating new category 'acl_
↪user_group'
15:45:19.959 [info] [example] info @ z_datamodel:169  Creating new category 'acl_
↪collaboration_group'
15:45:20.001 [info] [example] info @ z_datamodel:169  Creating new predicate
↪'hasusergroup'
15:45:20.033 [info] [example] info @ z_datamodel:169  Creating new predicate
↪'hascollabmember'
15:45:20.044 [info] [example] info @ z_datamodel:169  Creating new predicate
↪'hascollabmanager'
```

```
15:45:20.059 [info] [example] info @ z_datamodel:169  Creating new acl_user_group
↪'acl_user_group_anonymous'
15:45:20.071 [info] [example] info @ z_datamodel:169  Creating new acl_user_group
↪'acl_user_group_members'
15:45:20.079 [info] [example] info @ z_datamodel:169  Creating new acl_user_group
↪'acl_user_group_editors'
15:45:20.086 [info] [example] info @ z_datamodel:169  Creating new acl_user_group
↪'acl_user_group_managers'
15:45:20.393 [info] [example] info @ z_datamodel:169  Creating new category 'admin_
↪content_query'
======================== EUnit ========================
example_testone_sitetest: administrator_name_test (module 'example_administrator_
↪sitetest')...[0.022 s] ok
=======================================================
  Test passed.
ok
```

Deployment

So you have built your Zotonic site, and now you want to show it to the world. This page tells you how to configure your zotonic environment so that it is ready for real-world website visitors.

As per the Installation Instructions, up until now you probably have always started Zotonic using the `zotonic debug` command. This is fine for debugging purposes, because it gives you an Erlang shell in which you can view the output of the server, the `?DEBUG` messages that are triggered, and try out Erlang expressions.

However for a production system, you don't need this shell, you want Zotonic running in the background, and started at startup.

This manual describes the various ways how Zotonic can be run and how it works in combination with widely used HTTP frontends like Varnish and nginx.

Table of contents

Automatic startup on system boot

Once you have Zotonic running, you want to make sure that it automatically starts up when the server reboots, so that the website keeps running after a power failure, for example.

Creating an init script

The *Zotonic shell command* (page 496) can start Zotonic in the background and stop it again.

An init script will just need to call the zotonic command with either `start` or `stop`. On debian systems it might look like this:

```
#!/bin/sh -e

### BEGIN INIT INFO
# Provides:             zotonic
# Required-Start:       $local_fs $remote_fs $network $time postgresql
# Required-Stop:        $local_fs $remote_fs $network $time postgresql
# Should-Start:
# Should-Stop:
# Default-Start:        2 3 4 5
# Default-Stop:         0 1 6
# Short-Description:    Zotonic
### END INIT INFO
```

```
# Set any environment variables here, e.g.:
#export ZOTONIC_IP="127.0.0.1"

/usr/bin/sudo -u zotonic -i /home/zotonic/zotonic/bin/zotonic $@
```

For other environment variables which can be set, see *Useful environment variables* (page 72).

Server configuration

This chapter describes how to configure your Linux server for running Zotonic.

File descriptors

By default, Linux limits the number of file descriptors (i.e. the maximum number of open files) at 1024. Running Zotonic at scale will require many more than that, particularly because Zotonic uses *WebSockets* (page 365) extensively; remember that every open port is an open file.

The limit applies on several levels:

1. the process, in our case the Erlang VM (BEAM) that runs Zotonic
2. the Zotonic user
3. system-wide.

To check the current process usage, find BEAM's PID:

```
# pidof beam.smp
10006 (for instance)
```

Then count its open files:

```
# lsof -a -p 10006 | wc -l
```

And compare it to the process limit:

```
# cat /proc/10006/limits | grep 'Max open files'
```

You can raise the process limit by adding `ulimit -n` to your Zotonic init script:

```
ulimit -n 50000
```

Or change the limit in your system definition. For instance, when using systemd:

```
[Service]
LimitNOFILE=50000
```

Finally, make sure to check your system-wide limits, too.

Network connections

If you have stateful connection tracking enabled, high-traffic Zotonic sites may overflow your conntrack table.

Compare the current number of connections with the limit:

```
# sysctl net.netfilter.nf_conntrack_max
# sysctl net.netfilter.nf_conntrack_count
```

When you increase the maximum number of connections in the connection tracking table it is important to increase the size of the table which stores the connections. The rule of the thumb for this is: `nf_conntrack_buckets = nf_conntrack_max / 8`.

If you need to raise the limit, edit `/etc/sysctl.conf`:

Listing 2.1: /etc/sysctl.conf

```
net.netfilter.nf_conntrack_max = some_number
net.netfilter.nf_conntrack_buckets = some_other_number
```

Moreover, to reduce the number of open connections, you can decrease their time-out values so they get closed sooner. By default, inactive connections can stay in the conntrack table for 5(!) days. To change this:

Listing 2.2: /etc/sysctl.conf

```
net.netfilter.nf_conntrack_tcp_timeout_established = 600
```

If you have a proxy in front of Zotonic (e.g. HAProxy or Varnish), you need to change the limits on the proxy, too.

See also:

- See *Exometer metrics* (page 209) on how to monitor your Zotonic system.
- For general Erlang troubleshooting, Fred Hebert's free ebook Stuff goes bad: Erlang in anger is a very good resource.

Running on Port 80 and Port 443

Using standard ports helps visitors discover your page and removes the awkward port number from URLs.

The HTTP and HTTPS protocols are normally served on TCP ports 80 and 443. It is beneficial to run a these services on those standard ports: it aids discovery and lends a air of polish while a non-standard port number suggests something is incomplete.

*nix systems only allow the superuser (root) to bind to ports below 1024. HTTP & HTTPS use ports 80 & 443 respectively. So setting up Zotonic to serve from those ports without running as superuser presents a problem, as *nix considers all ports below 1024 to be "privileged", requiring special access.

Other webservers (like nginx) typically do not have this problem, as they usually run their main unix process as root, but forking off child processes as non-privileged workers. Zotonic cannot be made to work like that because it is just one unix process, and running Zotonic entirely as the root user is considered harmful.

This manual outlines three different methods to let Zotonic listen on port 80. All of them are for *nix based systems only.

Pre-launch notes

Before Zotonic serves its sites on a privileged ports, the hostname portions of your Zotonic sites need to be changed to reflect this.

For production release of your new Zotonic site you need to:

- Make sure that your server has a public IP address, and that the server accepts connections on port 80.
- For each of your Zotonic sites, configure their DNS (e.g. *www.mysite.com*) to point to your server's IP address.
- Change `{hostname, "mysite:8000"}` to `{hostname, "www.mysite.com"}` in `user/sites/mysite/config`. This last change enables the virtual hosting: it makes sure that Zotonic knows which site is being requested when somebody visits *www.mysite.com*.

Note: Your actual site location might be different, see the *User sites directory*.

Running behind another web server / proxy

You run another web server to proxy requests from port 80 to 8000 and from 443 to 8443. *Varnish* (page 67) and *nginx* (page 70) are both very capable web servers for doing this.

However, Zotonic itself is also very capable of directly serving web content without needing an extra caching layer in between. The other methods listed below explain how Zotonic can obtain direct access to the privileged ports.

Using authbind

Note: Instructions below assume site is named `mysite` and Zotonic is installed in `/home/zotonic/zotonic`. Replace as appropriate.

Install authbind:

```
zotonic:~$ sudo apt-get install authbind
```

Configure authbind to allow zotonic user to access port 80:

```
zotonic:~$ sudo touch /etc/authbind/byport/80
zotonic:~$ sudo chown zotonic /etc/authbind/byport/80
zotonic:~$ sudo chmod 500 /etc/authbind/byport/80
```

Set up the environment.

You need to tell Zotonic explicitly which IP address to listen on. Zotonic defaults to `any`, which will also bind to any ipv6 addresses available. However authbind doesn't work with ipv6 and so will cause Zotonic to crash on startup.

Add the following entries to `/home/zotonic/.profile`, then save file & exit:

```
export ZOTONIC_PORT=80
export ZOTONIC_SSL_PORT=443
public_interface=eth0
export ZOTONIC_IP=`/sbin/ifconfig $public_interface | grep 'inet addr:' | cut -d: -
↪f2 | awk '{ print $1}'`
export ERL="authbind --deep erl"
```

Where `eth0` is the name of the Ethernet interface that connects to the Internet. (For a MediaTemple (ve) host this was venet0:0) You could alternatively set it to `lo` for localhost-only testing or to a LAN-only interface (say eth1) for a multi-interface server you are using Zotonic to host an intranet site with.

Source the file to update the environment:

```
zotonic:~$ . ~/.profile
```

Delete the Zotonic config file (this will be re-generated automatically when zotonic next starts up):

```
zotonic:~$ rm ~/.zotonic/zotonic.config
```

Set the port for your site. Edit the hostname entry in `zotonic/user/sites/yoursite/config` to read as follows:

```
{hostname, "yoursite:80"}
```

Stop zotonic if already running:

```
zotonic:~$ ~/zotonic/bin/zotonic stop
```

Start zotonic:

```
zotonic:~$ ~/zotonic/bin/zotonic start
```

Browse to http://yoursite/ and verify that everything is working like it should.

Using setcap

Warning: this is a much broader approach as it grants privileged bind to all Erlang VM processes (the `beam` and `beam.smp` executables). Unless you are the sole user of such a machine this is not a great idea.

From a shell, install the setcap program:

```
sudo apt-get install libcap2-bin
```

Now configure setcap to allow Erlang BEAM processes user to bind to ports lower than 1024:

```
sudo setcap 'cap_net_bind_service=+ep' /usr/lib/erlang/erts-5.9.2/bin/beam
sudo setcap 'cap_net_bind_service=+ep' /usr/lib/erlang/erts-5.9.2/bin/beam.smp
```

Note that the exact paths to the `beam` and `beam.smp` can be different, depending on the Erlang version.

During package upgrades Erlang may be upgraded and your site will seem to be broken. Just make sure to check the ERTS version and rerun these setcaps commands for the new version.

For more granular control, you could create an Erlang release that only the Zotonic User can access. Once the release is created `setcap` could be applied to the beam and beam.smp within that release only.

Using iptables

If authbind and setcap will not work for you, using the system firewall to redirect the ports can be an option.

Firewall prerouting can be enabled as follows to forward communication on port 80 to port 8000 and port 443 to port 8443:

```
iptables -t nat -A PREROUTING -p tcp --dport 80 -j REDIRECT --to 8000
iptables -t nat -A PREROUTING -p tcp --dport 443 -j REDIRECT --to 8443
```

You also need two more rules so that the site can reach itself. In the following firewall rules, replace `your.ip.address` with your external IP address:

```
iptables -t nat -A OUTPUT -p tcp -d your.ip.address --dport 80 -j REDIRECT --to␣
↪8000
iptables -t nat -A OUTPUT -p tcp -d your.ip.address --dport 443 -j REDIRECT --to␣
↪8443
```

The downside of using the firewall is that Zotonic still also listens on port 8000. This might be a cause for confusion.

For instructions on how to save these firewall rules and reinstate them after a system reboot, consult the Ubuntu firewall administration manual.

HTTPS support

HTTPS support for Zotonic versions 0.9 and up is handled by *mod_ssl* (page 313). It allows each *Zotonic site* to configure HTTPS support, which runs on a different port for each site (due to the limitations of the HTTPS support in some browsers).

Using Varnish as frontend for Zotonic

Using the Varnish HTTP frontend, you can speed up your Zotonic even more as this web server caches static files intelligently.

Your Varnish `config.vcl` needs to define a *backend* for Zotonic:

```
backend zotonic {
  .host = "127.0.0.1";
  .port = "8000";
  .first_byte_timeout = 300s;
  .connect_timeout = 300s;
  .between_bytes_timeout = 300s;
}
```

Then, in `vcl_recv`, specify the Zotonic backend as the default backend:

```
sub vcl_recv {
  set req.http.X-Forwarded-Host = req.http.host;
  set req.backend   = zotonic;

  ...
```

Full varnish example configuration file

Please see the Varnish documentation for more information.

```
#
# Varnish configuration for a Zotonic site
#

backend zotonic_com {
  .host = "127.0.0.1";
  .port = "8000";
  .first_byte_timeout = 300s;
  .connect_timeout = 300s;
  .between_bytes_timeout = 300s;
}

backend nginx {
  .host = "127.0.0.1";
  .port = "8080";
  .first_byte_timeout = 300s;
  .connect_timeout = 300s;
  .between_bytes_timeout = 300s;
}


sub vcl_recv {
  set req.http.X-Forwarded-Host = req.http.host;

  ##########################################################################
  ############################ VIRTUAL HOSTS ###############################

  if (req.http.host ~ "^(www.|)(zotonic|example).(com|net)$") {
    set req.backend   = zotonic_com;
  }
  # Nginx hosted sites
  #
  else {
    set req.backend   = nginx;
```

```
  }

  ###########################################################################

  # Add a unique header containing the client address
  unset req.http.X-Forwarded-For;
  set   req.http.X-Forwarded-For = client.ip;

  # We only deal with GET and HEAD by default
  if (req.request != "GET" && req.request != "HEAD") {
    return (pass);
  }

  # Cache the home page for a short period (ttl = 1 second, see vcl_fetch)
  if (req.url ~ "^/$") {
    unset req.http.Cookie;
    unset req.http.Authenticate;
    set req.grace = 10s;
    return (lookup);
  }

  # Cache served css and media files
  if (req.url ~ "^/(lib|image|media|favicon.ico)/") {
    unset req.http.Cookie;
    unset req.http.Authenticate;
    set req.grace = 30m;
    return (lookup);
  }

  return (pass);
}


sub vcl_pipe {
    # Note that only the first request to the backend will have
    # X-Forwarded-For set.  If you use X-Forwarded-For and want to
    # have it set for all requests, make sure to have:
    set req.http.connection = "close";
    return (pipe);
}


sub vcl_pass {
  if (req.url ~ "^/comet") {
    #bereq.connect_timeout = 70;
    #bereq.first_byte_timeout = 70;
  }
  return (pass);
}


sub vcl_fetch {
  if (req.url ~ "^/(lib|image|media|favicon.ico)/") {
    unset beresp.http.Set-Cookie;
    set beresp.grace = 30m;
    set beresp.ttl = 10m;
    return (deliver);
  }
  return (pass);
}

sub vcl_error {
  if (obj.status == 503 && req.restarts < 4) {
```

```
    restart;
  }
}
```

Auto-starting Varnish on Mac OSX

To automatically start Varnish on Max OSX, add the following `plist` file to launchd.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/
↪PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>Disabled</key>
    <false/>
    <key>KeepAlive</key>
    <false/>
    <key>Debug</key>
    <false/>
    <key>Label</key>
    <string>varnishd</string>
    <key>OnDemand</key>
    <false/>
    <key>GroupName</key>
    <string>wheel</string>
    <key>UserName</key>
    <string>root</string>
    <key>ProgramArguments</key>
    <array>
        <string>/usr/local/sbin/varnishd</string>
        <string>-F</string>
        <string>-a</string>
        <string>:80</string>
        <string>-T</string>
        <string>localhost:6082</string>
        <string>-f</string>
        <string>/usr/local/etc/varnish/varnish.zotonic.vcl</string>
        <string>-s</string>
        <string>malloc</string>
        <string>-u</string>
        <string>nobody</string>
    </array>
    <key>RunAtLoad</key>
    <false/>
</dict>
</plist>
```

Proxying Zotonic with nginx

It is possible to put Zotonic behind the *nginx <http://nginx.org/>* web server, for example if you have other, non-Zotonic virtual hosts running on your system.

When proxying, don't forget to check the config files of the sites you are planning to server (the `user/sites/your_site/config` files). The `hostname` value should not contain any port number, if you run from port 80: `{hostname, "test.zotonic.com"}`.

Below is a configuration file we use to proxy nginx to zotonic. Be sure to replace all occurrences of `test.zotonic.com` with your own hostname:

```
server {
      listen 80;
      server_name  test.zotonic.com;

      access_log  /var/log/nginx/test.zotonic.com.access.log;
      error_log  /var/log/nginx/test.zotonic.com.error.log;

      keepalive_timeout 65;
      gzip off;

      location / {
          proxy_pass http://127.0.0.1:8000/;
          proxy_redirect off;

          proxy_set_header   Host             $host;
          proxy_set_header   X-Real-IP        $remote_addr;
          proxy_set_header   X-Forwarded-For  $proxy_add_x_forwarded_for;

          client_max_body_size       50m;
          client_body_buffer_size    128k;

          proxy_connect_timeout      90;
          proxy_send_timeout         90;
          proxy_read_timeout         90;

          proxy_buffer_size          4k;
          proxy_buffers              4 32k;
          proxy_busy_buffers_size    64k;
          proxy_temp_file_write_size 64k;
      }

      location /close-connection {
           keepalive_timeout 0;
           empty_gif;
      }
}
```

See the nginx documentation for more information on its configuration procedure.

Use nginx for SSL termination

It is possible to use nginx to terminate SSL. If this is done, then the *protocol* (page 492) configuration option needs to be set to `https`. Otherwise Zotonic doesn't know that the site is running on HTTPS.

This is an example configuration for nginx:

```
server {
    listen 443 ssl;
    listen [::]:443 ssl ipv6only=on;
    server_name hostname.tld;

    ssl_prefer_server_ciphers on;
    ssl_protocols TLSv1 TLSv1.1 TLSv1.2; # not possible to do exclusive
    ssl_ciphers 'ECDHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES128-GCM-SHA256:DHE-RSA-
↪AES256-GCM-SHA384:DHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-SHA384:ECDHE-RSA-
↪AES128-SHA256:ECDHE-RSA-AES256-SHA:ECDHE-RSA-AES128-SHA:DHE-RSA-AES256-
↪SHA256:DHE-RSA-AES128-SHA256:DHE-RSA-AES256-SHA:DHE-RSA-AES128-SHA:AES256-GCM-
↪SHA384:AES128-GCM-SHA256:AES256-SHA256:AES128-SHA256:AES256-SHA:AES128-SHA:HIGH:!
↪aNULL:!eNULL:!EXPORT:!DES:!MD5:!PSK:!RC4';
```

```
    add_header Strict-Transport-Security max-age=15768000; # six months

    ssl_session_cache shared:SSL:10m;
    ssl_session_timeout 5m;

    ssl_dhparam /etc/nginx/dhparam.pem;
    ssl_certificate /path/to/ssl.crt;
    ssl_certificate_key /path/to/ssl.key;

    location / {
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
        proxy_set_header Host $http_host;
        proxy_pass http://127.0.0.1:8000/;
    }
}
```

Useful environment variables

The following environment variables influence how Zotonic starts up.

ZOTONIC_IP Which IP address to bind the web server to. By default, it binds to any IP address. When running Zotonic behind a proxy like nginx or varnish, it is wise to put `127.0.0.1` here.

ZOTONIC_PORT The port number to bind the web server to. Defaults to port 8000.

ZOTONIC_SSL_PORT The port number to bind the ssl web server to. Defaults to port 8443.

ZOTONIC_SMTP_LISTEN_IP The IP address to bind the SMTP server to. Binds to any IP address by default.

ZOTONIC_SMTP_LISTEN_PORT The port number to bind the SMTP server to. Defaults to port 2525.

ZOTONIC_SMTP_LISTEN_DOMAIN The domain to bind the SMTP server to, if any.

TMP Where Zotonic puts temporary files, like temporary files for image resizing or URL downloading.

Note: When the variables do not seem to have effect, check `~/.zotonic/zotonic.config`. The above variables are overridden by the ones in that file.

Contributing to Zotonic

We encourage contributions to Zotonic from the community! This chapter describes how you can help improve Zotonic.

1. Fork the `zotonic` repository on Github (at https://github.com/zotonic/zotonic).
2. Clone your fork or add the remote if you already have a clone of the repository:

   ```
   git clone git@github.com:yourusername/zotonic.git
   ```

   or:

   ```
   git remote add mine git@github.com:yourusername/zotonic.git
   ```

3. Create a topic branch for your change:

   ```
   git checkout -b some-topic-branch
   ```

4. Make your change and commit. Use a clear and descriptive commit message, spanning multiple lines if detailed explanation is needed.

5. Push to your fork of the repository and then send a pull-request through Github:

```
git push mine some-topic-branch
```

6. A Zotonic committer will review your patch and merge it into the main repository or send you feedback. The pull request page on github is the main place to discuss the code and related topics.

Coding standards

Note: As these conventions were established only after a large part of the code base has been written, the code style described here is not yet in effect in all parts of the Zotonic code base. We're trying to gradually adapt to it, however.

When writing code, do not introduce trailing whitespace and try to keep lines of code shorter than 80 characters.

Provided with the Zotonic distribution is a Zotonic template mode, `zotonic-tpl-mode`, which supports the Zotonic flavor of ErlyDtl. It is located in the `priv/emacs/zotonic-tpl-mode.el` file, and may be installed in emacs by adding something like this to your *.emacs* file:

```
(add-to-list 'load-path "../path/to/zotonic/priv/emacs")
(require 'zotonic-tpl-mode)
;; optional, for associating .tpl files with zotonic-tpl-mode
(add-to-list 'auto-mode-alist '("\\.tpl$" . zotonic-tpl-mode))
```

Indents

We use the "Emacs-style" indenting (using `erlang-mode` provided with the Erlang distribution). This indenting style seems to be a convention in much of the Erlang world.

The indent is 4 spaces (no tabs).

Writing commit messages

The Zotonic commit convention are slightly based on rebar's README.

Structure your commit message like this:

```
prefix: One line summary (less than 50 characters)

Longer description, multiline text that is supposed to wrap at 72
characters.

Fixes #403
```

- **Prefix**: Every commit message must start with one of the designated commit prefixes:
- `mod_foobar:` Changes that are related to a single module should be prefixed with the module name.
- `doc:` For changes to the documentation, everything below doc/
- `scripts:` for changes to the `zotonic` command and its helper scripts.
- `build:` for the build system and related changes.
- `tests:` for unit tests and the testsandbox.
- `skel:` for the skeleton sites.
- `zotonic_status:` for the default site.
- `translation:` for new/updated translations.

- `core:` For changes in the *src*, *include* or *deps* folder; e.g. anything not covered by another tag.
- The **summary** should be less than 50 characters, and tell what was changed. Use the imperative present tense (fix, add, change). For example: *Add 'foobar' filter*, *Fix bug in media upload service*.
- The **description** should explain the intention and implementation of your approach, in the present tense.
- Optionally, when your commit **fixes** a bug on github, add *Fixes #1545* on a separate line below the description.

Notice the empty line preceding the longer description and the "Fixes" tag.

Git best practices

- Please maintain commit atomicity by breaking up logical changes into separate commits; e.g., do not commit unrelated fixes into a single commit.
- Make whitespace changes separately.
- When updating from the Zotonic source, please use `git pull --rebase` to prevent unnecessary merge commits.
- Generally, try to Mind your Git Manners.

The CONTRIBUTORS file

When this is your first contribution to Zotonic, you are welcome to add your name and e-mail address to the CONTRIBUTORS file in the root of the project. Please keep the file alphabetically ordered.

Running the tests

Zotonic comes with a basic test suite which can be run the following way:

```
zotonic runtests
```

This starts the Zotonic system and executes all EUnit tests. It will disable all zotonic sites except for the special site `testsandbox`, which will be enabled.

The `testsandbox` site does not have a database configuration and is configured to run on `localhost:8040`.

Contributing documentation

Build the documentation

First, install Sphinx. To build the documentation, Erlang must be installed.

```
$ cd doc/

# Install dependencies
$ pip install -r requirements.txt

# Generate meta-*.rst files:
$ make stubs

# Then generate HTML files:
$ make html
```

Then view the HTML files in `doc/_build/html/index.html`.

Heading styles

Use the following convention for headings:

```
First-level heading
===================

Second-level heading
--------------------

Third-level heading
^^^^^^^^^^^^^^^^^^^

Fourth-level heading
""""""""""""""""""""
```

When writing documentation of modules, actions, etc.; anything under `ref/`; the first level heading is already there for you, generated in the `meta-*.rst` file. So you should only use `----------` and `..........` for the headings in the `ref/` files.

When using Emacs, this little snippet helps with adding underlines to headings:

```
(defun underline-with-char (char)
  (interactive (list (read-from-minibuffer "Char: ")))
  (when (= 0 (length char))
    (error "Need a character"))
  (setq char (aref char 0))             ; Ignore everything but the first char.
  (save-excursion
    (goto-char (point-at-eol))
    (insert "\n"
            (make-string (- (point-at-eol)
                            (point-at-bol))
                         char))))
```

From a mailing list post.

References

Be generous with using references (`:ref:`pagelabel``) in your writing. The more terms are linked to their respective documentation pages, the better. Only make the first occurrence of a term a reference to its page, though; consequent occurrences can be made `` `italic` ``.

Add a `.. seealso::` section at the bottom to highlight any other pages which are closely related to the current one, for example:

```
.. code-block:: none
```

See also:

Contributing to Zotonic (page 72)

Table styles

For the easy editing of tables, we use Emacs' table-mode, which at first has a bit of a learning curve but actually works pretty well when creating the ascii-art tables that the RST format requires you to use.

In general, we use this style of table:

```
+--------------------+--------------------+
| Header             |Other header        |
```

```
+====================+====================+
|This is the table   |Some more contents  |
|cell contents       |                    |
+--------------------+--------------------+
```

Writing consistent Cookbook items

A Zotonic Cookbook item is a single-concept solution to a well-defined problem, living in the *Cookbooks* (page 197) section of the documentation.

Useful items range from the simplest content management tasks to technically sophisticated module development and site administration solutions. This means that items are welcomed from noobies and wizards alike.

Whenever you struggle to find a solution to a specific problem, fail to find a Cookbook item that addresses it, and work through the solution with a final "Aha!," you have the raw material for an excellent Cookbook submission.

A well-written item has four sections:

WHY: What problem does this Cookbook item solve? What benefits does it deliver?

Four major reasons for submitting Cookbook items are:

1. The best way to learn is to teach
2. Your Cookbook items documents your efforts; helps you remember what you did next time you encounter a similar problem
3. Each item makes it that much easier for noobies and other community members to advance their Zotonic skills.

ASSUMPTIONS: What does this item assume about operating system, Linux distribution, programming skills, knowledge of Zotonic architecture and conventions etc.

HOW: Step-by-step instructions for implementing your solution.

Don't take user competency for granted. When you specify a command, note what user name you're working under and what directory you are working in. Respect the noobies by including steps that may be obvious to you but not so obvious to folks with less experience.

Think of your instructions as a check-list. A noobie should be able to achieve success by reading, implementing and checking off each instruction. Keep your instructions simple, complete, and clear.

Recruit a noobie to try out your solution. Fix the stumbling blocks s/he encounters. If you can't find a noobie, put yourself in noobie mind. Remember, you too once were one.

Zotonic releases

Release dates

Zotonic follows a time-based release model. Every first Monday of the month – at the call of the Dutch test siren – a new Zotonic version is released.

Release schedule

Preparation for each release lasts one month:

1. **Development phase**: new features are added and existing ones improved. Commits take place on the current development branch (for instance, `0.x`).
2. **Stabilisation phase**: five working days before a release, we create a release branch (for instance, `release-0.16.0`) from the development branch. During the stabilisation phase, no new features are added. Instead, the last bug fixes for the release are committed.

3. On the first Monday of each month, the release branch is **tagged** (for instance, `0.16.0`), merged back into the development branch and then discarded.

See also:

GitHub for the latest release.

Release Notes

Every time a Zotonic release is made, we create a document which lists the most important changes. This page contains links to the release notes for each release so far, dating back to Zotonic's initial release in November 2009.

Release 0.1.0

Released on 2009-11-13.

- Initial release.
- Packaged the zotonic.com site as the prime "example" site in the default install.

Release 0.2.0

Released on 2009-12-11.

New modules

mod_broadcast Send system messages to all users which are currently logged in in the Zotonic admin.

mod_calendar Shows event resources in a week-overview, and generates ICalendar feeds.

mod_mailinglist Module which allows you to define groups of recipients and send mailings to them. Can also send via the unix sendmail program.

mod_twitter Receives feeds for Zotonic persons,using the Twitter streaming API.

- New core features:

"catinclude" and "all catinclude" tags These include templates based on the category of a resource. Used in the admin to create custom fields based on category. http://zotonic.com/documentation/760/catinclude

Query search model Generate lists of resources on the fly. Used in mod_atom_feed to generate atom feeds, and has an API entrypoint, /api/search. http://zotonic.com/documentation/761/the-query-search-model

More template filters: in_future, in_past, rand, twitter, escape_ical

Bugfixes

- Dynamic postgresql pool size, based on system load (issue #4)
- Issue in postgres pooling on stresstesting (#15)
- Uploaded files now get a proper mime type and extension (#5)
- And other issues: #2, #3, #9, #11, #14, #19, #20

Release 0.3.0

Released on 2010-01-25.

New modules

mod_comment Enables a simple commenting system on your site using mod_comment.

New core features

A new default site The default site of a vanilla Zotonic install is now modelled after a simple blog-style website, complete with an archive section, keywords, navigation to previous and next posts, atom feeds and comments.

Speed improvements The Webmachine code was restructured to be more lean-and-mean, yielding up to 20% more performance on page requests.

WebSockets support When WebSockets is available in the browser, then it is used as a replacement for the Comet long poll. Currently only Google Chrome supports this feature but it is expected to arrive in other browsers soon.

Admin updates Support for editing a location (Google map picker), a new collection type "query" was added for creating "saved searches".

EUnit support A start has been made to put the core functionality of Zotonic in unit tests using the EUnit testing framework. As of yet, only a small fraction of the code has been covered, but we'll keep working on increasing the code coverage of the tests.

Bugfixes

- Resizing animated GIFs (#28)
- Determining EXIF orientation for images (#27)
- The OAuth API key management interface is now available from the admin. (#35)
- Hiding "meta" pages from the admin overview (#12)
- And dozens of small fixes which did not go through the issue tracker.

Release 0.4.0

Released on 2010-04-19.

New modules

mod_pubsub Enables resource sharing over XMPP's PubSub; share content between sites and get realtime updates when content changes. See: http://scherpenisse.net/id/644

mod_search_solr Added a module which plugs into Zotonic's search system to support Solr (http://lucene.apache.org/solr/). Using Solr enables quick fulltext searching and facetting.

New features

Default site improvements The default site of a vanilla Zotonic install has been improved with nicer graphics, cleaner typography, a "contact form" example and styles for the Twitter module.

"More results" scomp A twitter/facebook style ajaxified "read more" pager, which is a button which will fetch more results for the current search question inline on the same page.

Windows support Initial support for building and running Zotonic on the Windows platform.

Database schema support Multiple sites running inside one Postgres database is now possible thanks to Postgres' support for multiple table namespaces (schema's)

Template expressions It is now possible to use full boolean and arithmetic expressions in the ErlyDTL templates.

Other features:

- Webserver IPv6 support
- Yandex.Video support in mod_video_embed module (#52)
- PID-file for zotonic (#74)
- Support for HTML5 audio/video tags in TinyMCE editor (#75)
- Newer TinyMCE 3.3.2 release from upstream (#69)
- Newer Mochiweb r153 release from upstream

Bugfixes

- page_path controller should not redirect to the default_page_url (#6)
- Get the name of the current dispatch rule (#21)
- zotonic fails after postgresql restart (#49)
- Unreliable pivot? (#50)
- Module manager should feedback when module cannot be started. (#51)
- Do not depend on the 'default' site (#59)
- i18n of scomp_pager (#62)
- Buttons and "Logoff" link problems in Chrome (#63)
- Comment form breaks on new default site (#64)
- Getting an unknown_rsc error on startup (#66)
- Zotonic fails to (re)start if an existing admin panel is open with browser supporting WebSockets (#70)
- can't save location without e-mail (#71)
- Improve the default styles to include list bullets/numbers (#72)
- Twitter module cannot be enabled (#76)

Release 0.5.0

Released on 2010-10-03.

New features

Simpler module system Modules are simpler, do not have to be a fullblown gen_server. Registering z_notifier for modules is made more simpler by using Erlang's introspection on modules.

i18n support through gettext Gettext .po and .pot file support for translations. Templates can be translated per module. Pot files are automatically generated from the templates.

Pluggable Access Control system The new ACL structure works through pluggable ACL modules. Two ACL modules are included as examples. mod_acl_adminonly, where all users are admins, and mod_acl_simple_roles, which implements a simple role based ACL system.

Authentication can now be customized and extended. mod_authentication is the basic module used for authentication. This module can be extended. The mod_facebook is an (incomplete) example of such an extender. mod_authentication implements the username/password authentication, including logon and logoff. It also supports 'password forgotten' e-mails.

User signup Non admin users can sign up using the mod_signup. This module works in harmony with the authentication module and authentication extenders.

New OTP supervisor hierarchy. The PostgreSQL connection pool is now part of the individual sites. Sites are more isolated and can be individually started, restarted or stopped. It is possible to add and remove sites without restarting Zotonic. Modules are now isolated and the running status of a module is displayed in the admin's module overview.

A status overview site, zotonic_status. zotonic_status shows the running status of all sites. When logged in, the user can start/stop/restart sites using his browser. It is also possible to do 'hg pull' updates of sites that contain a mercurial repo.

New ErlyDTL filters group_by_title_firstchar, is_visible, pprint, urlize, without_embedded_media.

Media preview enhancements {% image %} now supports the the following new arguments:

'extent' - create a larger image then the original not by scaling up but by adding a border to the image.

'removebg' - removes the image's background. It accepts an optional fuzziness parameter (range 0..100).

'upscale' - Forces a small image to scale up to the requested dimensions.

Extended support for Websocket connections. The two newest protocols, as used by Chrome and Safari, are supported.

mod_development improvements It now supports turning on or off the concatenation of {% lib %} includes as one file or separate files, and can give a live trace of translated templates, showing clearly the template inheritance and selections.

mod_menu improvements It implements the menu now as a template, easing your own menu implementation.

mod_emailer improvements It can now inline images into the e-mails

New: mod_slideshow It can make a slideshow of any collection, you can add your own slide templates.

New: mod_contact Simple contact form which gets sent over e-mail

New: mod_facebook Facebook logon

New: mod_imageclipper A simple javascript image-clipper bookmarklet for grabbing images from other websites.

New: mod_logging A realtime log of debug messages and errors in the system.

Other features:

- New ErlyDTL tags: {% inherit %}, {% overrule %}
- New ErlyDTL support for multiple argument {% with %}: {% with a,b as c,d %}
- New ErlyDTL support for filters with multiple parameters.
- New ErlyDTL test set, including regression tests.
- System wide configuration system (z_config) using a configuration file at 'priv/config'

Bugfixes

- Allow HTML5 audio and video tags (#75)
- Typo in m_config, line 127. undefind -> undefined (#83)
- setting initial admin password does not work (#88)
- After upgrading the code to latest changeset admin authentication causes exception (#91)

- Menu module does not follow ACL rules (#92)
- Crash in start.sh using Erlang R14A on Mac OS X 10.6 (#93)
- Extra Atom Link (#95)
- Makefiles use rm GNUism (#96)
- z_email:split_name_email/1 does not what it says it should do (#97)
- dots in page paths are transformed into dashes (#98)
- attaching media to pages does not work correctly (#99)
- After a module crashes, the new dynamic observe_* methods are not re-initialized (#100)
- setting page path and unique name is broken (#101)
- IF statements on empty rsc_list structures (#104)
- When image is too small, providing only a width should not make the image very large (#105)
- And many various other fixes which users noted on the mailinglist and were fixed quickly.

Release 0.6.0

Released on 2011-02-12.

New features

SSL support Zotonic has gotten support for serving web pages over secure HTTPS connections. When configured, it listens by default on port 8443. See http://zotonic.com/https-support for details.

z_logger A new subsystem for the low-level logging and tracing of requests. This module should be used to log lower level events during development time. Higher-level log messages (e.g. events by Zotonic modules) are still handled by 'mod_logging'.

multilingual content Every resource can have be translated in as many languages as you like. The admin has gotten an interface to provide the editing of the multiple language versions. Available languages are fully dynamically configurable.

z_depcache Partial rewrite of depcache system, is now faster and using more the process dictionary of the calling process to cache often used values.

New and changed modules

mod_signal New module providing a handy signal and slot mechanism for use in templates.

mod_tkvstore New module providing a simple typed key/value store for modules and Erlang code.

mod_translation Check if the user has a preferred language (in the user's persistent data). If not then check the accept-language header (if any) against the available languages.

mod_mailinglist Tweaks in the templates, updated dutch translations; do not send mail when deleting recipient from admin; Added 'recipient_id' to some e-mails so that the e-mails are sent in the correct language.

mod_authentication Fix user name display in password reminder e-mail.

mod_emailer Fix for e-mail override, escape the '@' in the original e-mail address. Added flushing of poll messages

mod_seo Added option to set a no-index for a complete site. New Google Analytics tracker code. With thanks to Richard Fergie.

mod_contact Configurable from address for contact email

mod_admin_identity Fix for finding users, select only identity records with type 'username_pw'

mod_calendar Better handling for undefined date_end values.

mod_search Improper months ordering in archive_year_month query. (#134)

mod_menu Possibility to create an arbitrary number of different menu's. Also a new filter (menu_trail) which gets the menu trail for the main menu.

Changes to template filters and tags

'first' filter added optional length parameter

min/max and minmax 3 new filters were added to clamp a value in an (integer) range.

filesizeformat New filter, similar to the Django filesizeformat filter.

lib tag Extended the lib tag with a 'use_absolute_url' option.

confirm/alert actions These actions were changed and now use HTML dialogs instead of javascript popups.

reversed New filter to reverse a list

menu tag Added 'menu_id' parameter to specify which menu to render

date_diff New filter to calculate the difference between two dates

tinymce_add, tinymce_remove New actions to dynamically initialize of de-initialize rich textareas

trigger_event New action to trigger a named wire.

wire Added a new 'visible' wire type, which triggers when the wired element comes into view (by scrolling or using 'show').

lazy New scomp which shows a 'loader' image and performs onetime actions when loader comes into view.

General bug fixes

- Fix for 'double-dot' in e-mails when using postfix. Also encode the $. characters using quoted-printable.
- Fix for format_price filter. Show thousands when no cents.
- Make video embed code editable.
- Merged various webmachine fixes, updating it to 1.7.3:
- support {stream, TotalSize, StreamFun} body result for range-capable streams
- Add infinity timeout to gen_server calls
- Allow multiple IP/port bindings
- split chunk header on semicolon just in case a client is using chunk extensions
- properly extract peername from all rfc1918 addrs
- change H7 to match on any if-match, not just *
- webmachine: WM-1.7.3(compat) ignores client's Content-Type on HTTP PUT requests (#130)
- webmachine: prevent using chunked transfer encoding with HTTP/1.0.
- increase the startup timeouts for the gen_servers to prevent startup race condition
- Update mochiweb to latest version from mochi/mochiweb github repository (1.5.0)
- Pulled latest epgsql driver to support Postgres notifications.
- Added additional mime types (Office 2007, .rar)
- z_session: Only mark the persistent store as dirty when a persistent value changes.

- pgsql: Fix for a problem where a postgres connection was not returned to the pool in case of a sql error.
- z_media_preview: some files without a preview where not showing an icon.
- fixed an DoS vulnerability in Mochiweb/SSL
- Added flushing for most periodic internal messages (e.g. tick, poll)
- windows: fix build.cmd; remove some unix-specificness from imagemagick shell commands
- mochiweb: Cookie expire date format string now follows rfc2109
- ACL checks on static file serving
- Comet: support for cross-domain comet connections

Release 0.6.1

Released on 2011-07-06.

Konstantin Nikiforov

- jquery actions insert_before, after, replace

Atilla Erdődi

- Activity Streams
- added notification for comments

Arjan Scherpenisse

- Mailinglist - fix database updater
- Fixed zotonic-stop command by using -name instead of -sname
- Services API tweaks
- Made group_title_firstchar filter work with translatable content.
- Fix up base/info service – add translation lookup to administrator title.
- Small tweak to atom entries, better follow atom spec
- Remove link to atom_feed from mod_base's base template to remove dependency on mod_atom_feed.
- Fix lib tag for including multiple files directly from the toplevel 'lib' folder.
- Release depcache locks immediately after exception in z:memo().
- Use translation lookup in internal link action.
- Make dispatch rule reloading more stable.
- Make z_utils:url_decode/1 work and add unit tests.
- z_media_tag: Ensure that the generated URL is a binary, even if it's empty.
- Fix for editing content which used to be multilingual but is no longer multilingual.
- backport changeset 05a5254b6c92 - Fix for embedded media not showing up
- Fix translation statistics for modules in places other than modules/
- Fix month/day name capitalization: dutch does not do capitalization.
- Fix TinyMCE double initialization for the rsc-body.
- Fix unnecessary sharpen in lossless (PNG) images.
- Backported fix for issue 158 to 0.6 (twitter images)
- Fix solr searching because z_convert:to_isotime now returns a binary.
- Allow for recursive file lookups in z_module_indexer:scan_subdir/4

- Fix menu fetching on 0.6; update docs
- Fixed essential typo in mailinglist module.
- Only use gzip encoding for mime types that actually benefit from it.
- Compile the Zotonic behaviours before the other source files.
- Fix mailinglist upload button for changed #upload{} record.
- Fixed a bug in mod_acl_simple_roles which caused a NULL violation when setting visible_for.
- Make zotonic-1.0.js compatible with jQuery 1.5
- Remove string reversal from mod_logging which was previously needed.
- fieldreplace: Removed toggle class from click handler, because change handler is also triggered.

Marc Worrell

- Show the 'voorvoegsel' field for nl, de languages.
- Add identify/2 for #upload records.
- Small style fix for non-translation editing of the body.
- Added support for subscribing persons in the database to a mailinglist.
- Added flipchart mime type.
- Fixes for looking up flipchart, upper case extensions and mime icons.
- Fix for getValue on an empty select element.
- Make pivoting dates more resilient against illegal dates.
- Fetch the widget options from data-... attributes instead from in the element's class. The class is still parsed for backwards compatibility.
- Move widget options to data-... attributes.
- Remove newlines before parsing the widget options.
- always call the widgetManager when inserting content.
- Added 'visible' option, triggers the moreresults action on visible of the element instead of on click.
- Added z_comet to the list of zotonic query args.
- Added 'if' filter.
- Added missing mailinglist footer template.
- Changes in the acl_add_sql_check notification, now uses a record, including the search sql.
- Added visibility toggle to mod_comment.
- Added support for generic msoffice mime type returned by 'file'
- Fix for resetting the textarea after committing the comment.
- Clear textarea and input fields after a comment has been posted.
- Added randomize filter, shuffles a list.
- Added 'matching' type of question and results view for select in 'narrative'
- Wired the 'postback_notify' event directly into the postback/websockets resources. This makes javascript triggered postbacks more flexible and simpler.
- Allow data_xxxx attributes, which map to html5 'data-xxxx' attributes.
- Added 'do_slideshow' widget. Needs a data-slideshow with the id of the elements to be shown in a slideshow dialog.
- Added cursor property for do_slideshow elements.

- Fix for validator, empty string should pass the format validation.
- Fix for black background when previewing transparent images as jpeg.
- Added form reset action. Fix for submit/reset of 'closest' form. Fix for IE problem with inputoverlay.
- Let do_inputoverlay also work for textareas
- Added 'propagate' option to event action (wire scomp). This prevents canceling the event after being handled.
- Added created column to overview.
- Better embedding of images. Now fetches the image using the rights of the sender. Also initialises the random number generator for the inline image cids.
- Removed content disposition of 'inline' images.
- Add X-Attachment-Id to embedded images.
- Added plain text alternative to sent e-mails. Based on markdown text of the html.
- Added 'acl' option, similar to the acl option of the resource_template.
- Better feedback when a validation fails on form submit.
- Added 'is_public' and 'ongoing' query terms.
- Added z_template:render/2 to render a #render{} record.
- Added poll, including poll templates. A poll should have a start/end date, used for displaying.
- More subtle error message when form does not validate.
- Fix for a problem where the substr of translated binaries was taken.
- Fix for binary format string.
- Fix for binary results from translations.
- Made do_listfilter more generic.
- Make erlydtl_dateformat a binary. Correct handling of binaries or lists for day/month representations. Re-enable test set.
- Allow institutions to have an username/password.
- Show date range for surveys. Always show poll results.
- Let the 'remeber me' be valid for 10 years or so instead of two weeks.
- Fix for max_age of persist cookie
- Updated modernizr, fixes hanging bug in Chrome.
- Added survey result download in CSV (tab delimited) format.
- Allow to specify 'propagate' with the wire scomp
- Added download link to survey results. Also changed tab delimited results file to comma delimited file.
- Hide results when user is not allowed to see the results.
- Fix for editing multilingual content with or without mod_translation.
- Added possibility to have 'undefined' array values.
- Log an error when we run into a compile error.
- Simple expression evaluator, makes use of the erlydtl parser and runtimes.
- Added conditional jumps on page breaks. Added missing license texts.
- Added fix for 'file' utility identification error on mpeg files.

- Added l10n module for further localization. Currently implements a list of languages by iso code with translations.
- Added a 'whereis' api to z_module_manager to find the pid of a running module.
- Fix for mod_l10n description.
- Fixes for pivoting country names. Expands iso code to country names in languages as defined in the page. Only use the english postgresql indexing to prevent problems when searching in other languages.
- Map upper case iso to lower case for pivoting.
- Workaround for validations of elements that are removed. Needs better solution.
- Fixes for dynamic changes in LiveValidation checked forms. Keep the administration of LiveValidation objects in the data() of the form elements. So that you can dynamically add/remove elements without the LiveValidationForm object getting out of sync.
- Prevent retry loop when providing an illegal page id
- Move the init of non-gen_server modules to the dummy module process instead of the module manager's process.
- Added user_from_page/1 and user_from_session/1 to derive the user id from a page or session pid.
- Fix for a problem where the 'appear' was called on the enclosing container for insert_top/bottom instead of the inserted html fragment.
- Part of previous commit to fetch user_id from page pid.
- Added do_popupwindow widget to open a link in a popup window.
- Replace textual smiley codes with images.
- Supervisor for named processes, based on Erlang's supervisor.erl
- Added firefox smiley.
- Fix for a problem where widgetManager is only called for the first element in the jquery context.
- Fix for popupwindow namespace.
- Fixes for window name on IE, and fixes for correcting size of popup after opening window.
- Delayed size correction, Chrome returns 0 when the window is just opened.
- Added support for anchors and tables to markdown translators.
- Added no_html option to html2markdown filter.
- Make it possible to pass additional variables to the rendered result template.
- Use the no_html option when translating html to plain text.
- Fix for z_render:set_value/3
- Allow property names as string values.
- Added cc to email record.
- Added support for Cc
- let os_escape of undefined return []
- Fix translation table references when pivoting resources.
- Added 'full' option for popup windows.
- Added live updating do_timesince. Added z_translate function, uses a lookup object to find translations. Fixed local/utc time problem in 'U' date format. Fixed copyright name in smiley.
- Fix for site/hostname mixup.
- Fix for opening a window in 'full screen' size.

- Echo q.username so that we can have direct link to the password reminder when we know the username.
- Added version to activity record.
- Added missing survey_results resource.
- Added notifications on edge insert/delete. Incomplete but good enough.
- Made session timeout configurable, use site.session_expire_1 and site.session_expire_n
- Automatically cleanup temporary files when the request process is stopped.
- Allow more than one `do_` widget per element.
- Fix for resizing images embedded in e-mails.
- Added Google Analytics _trackEvent do_gaq_track
- Added m.comment.get for a comment_id
- Added reload of pages on logon or logoff - first part of smarter session management and open pages.
- Let the reload action take into account the continuation page of a logon screen.
- Merging 0.7 email additions to help with site transitions from 0.6 to 0.7
- Added to_flatlist/1
- Added 'attachments' field to email record, for easing upgrading 0.6 sites to 0.7.
- Added newer jquery and jquery ui to fix drag problems in IE9
- Remove <style> tags when converting html to markdown.

Release 0.7.0

Released on 2011-07-28.

New core features

SMTP Native SMTP support for sending and receiving e-mails in any Zotonic site. We integrated Andrew Thompson's gen_smtp library which allows us to manage outgoing ánd incoming mails. mod_logging provides a new email log-view for inspecting what mails go in and out.

Commandline A "zotonic" shell command. The "zotonic.sh" shell command has been replaced by a more generic and more powerful shell command with support for pluggable subcommands.

Module repository Zotonic now supports installing system-wide modules which are not part of the core repository. We have created a place where externally contributed modules can be linked at http://modules.zotonic.com/. Modules registered on that site can be easily installed through the "addsite" subcommand.

Skeleton sites The default website has been replaced by the notion of "skeleton" sites. The "zotonic addsite" command lets you create a new Zotonic website based on one of the (currently two) website templates.

New modules

mod_email_relay Relay received e-mails to an user's email address. Serving as an example for the SMTP functionality, this module looks up a username by the local part of a received e-mail and forwards the mail to the mail address the user configured.

mod_email_receive Handle received e-mails, notifies email observers depending on a stored mapping of recipient addresses.

mod_import_csv Fairly generic module for importing CSV files, updating or creating new content on the fly.

mod_import_wordpress Basic import module for Wordpress WXR file format, allowing you to migrate a Wordpress blog into Zotonic.

Discontinued modules

To make Zotonic more lightweight and remove some of the build dependencies, some infrequently used modules have been removed from the core and moved to their own repository, at http://code.google.com/p/zotonic-modules/. These modules are mod_search_solr, mod_pubsub, mod_slideshow, mod_broadcast, mod_imageclipper, mod_admin_event and mod_calendar. They can still be easily installed with the help of the "zotonic modules install" command. The mod_emailer module (and its esmtp library) has been removed in favor of the native SMTP sending/receiving capabilities.

Each module now also can have its own dependencies by including a "deps" subfolder in the module. This is used for example in the mod_pubsub external module which has the exmpp library as a dependency.

Other minor features

- to_json filter for representing template values as JSON objects
- split filter for splitting a string
- slice filter for manipulating lists
- Added {% raw %}..{% endraw %} support for representing literal code blocks.
- erlydtl: Added possibility to define atoms using backquoted strings.
- admin templates are reorganized, allowing to write admin customizations with less code
- translations of the admin updated and more translations added

Bugfixes

Too many bugfixes to list. However, the base system is becoming more stable and this release aims to be a good step towards the 1.0.

Release 0.7.1

Released on 2011-08-03 13:17 by arjan.

Arjan

- Documentation fixes and fixed wrapping of homepage links in the www.zotonic.com site
- Fixed the menu editor: Added resource_menu_admin_menu again which was removed by accident.
- Reworked the way mod_logging notifies the log pages; it now uses mod_signal.
- Fixed quality=xx parameter to {% image %}.
- Added mod_signal to the default list of installed modules.

Release 0.7.2

Released on 2011-12-11 19:51 by arjan.

Alain O'Dea (1):

- Ignore compile output and runtime generated files

Arjan Scherpenisse (mod_mailinglist):

- Added new bounce handling dialog.
- Attach documents to the mailing for each 'hasdocument' edge instead of 'document'.
- Added administration of worker processes to email server.
- Updated translations, removed references to mod_emailer.
- Fixed using the configured address in mailinglist rsc as the "from" address.
- Re-added the option of test-sending a mailinglist page to a single address.
- Removed already_sent check from mod_mailinglist, which is not needed since the new interface.
- Fix include reference to mailing footer template.

Arjan Scherpenisse (mod_survey):

- mod_survey: made questions configurably required or not.
- Added dutch translation for mod_survey; small template tweaks. Fixes #205
- mod_survey: quick hack to put email validation on a field if you name it 'email'.

Arjan Scherpenisse (misc):

- Added support for varying overview lists in the admin on category.
- mod_facebook: make 'scope' parameter configurable.
- On win32, mime type returned as application/octet for all files.
- Added spanish translation and install the spanish language by default and enable it.
- Added a file with the translators per language.
- Scan all erlang files in the modules for translatable strings.
- Added missing file
- Added 2 new translation template files.
- Add check in zotonic-addsite to see if the given site name is valid.
- tiny_mce: Fixed the disappearing of inline images. Fixes issue #203.
- Tooltip Y position is now dependent on its height. Fixes issue #207
- Disregard stderr output from identify command. Fixes issue #206
- Make windows users happy when redirecting stderr.
- Added dutch month/day names to mod_l10n.
- z_datamodel: do not try to resolve 'undefined' in valid_for check
- Generalized group_title_firstchar filter into group_firstchar.
- Added date_start_year= and date_end_year= search filters to filter on year of date start/end.
- Fix infinite recursion in sub_month/3 filter.
- mod_import_csv: Added import button to admin status page.
- m_rsc_update emptied the pivot_date_* fields when `date_` fields where not part of the update.
- Added application/x-font-woff for webfonts.
- Allow modules to override admin TinyMCE options which were originally set in admin-common.js
- Addressed the issues in the backup module. Fixes #220
- mod_backup: make sure we have an archive dir before archiving.
- Added option email_bounce_override to override bounce email address.
- Allow id to be either number or unique name for resource_admin_edit.

- Export z_session_manager:get_session_id/1.
- z_html: do not escape/strip HTML when a property name ends in _html.
- z_html:escape_props/1 - Make selecting escape function more safe.
- Fixed picture rotation detection by tweaking the parser of the output of "exif -m -t Rotation".
- Fixed z_utils:tempfile() to respect OS environment variables.
- z_convert:to_json/1 now also accepts floating point numbers.
- Fix SQL error in z_installer.
- Add distinctive icon for internal link in TinyMCE. Fixes #189
- Removed m_identity:{get,set}_props which were unused and not working.
- Show language selector on admin media page. Fixes #253
- mod_twitter: remove invalid {verbose, trace} option.
- lower/upper filters now try to convert their argument to a list if they're not.

Release 0.7.3

Released on 2011-12-14 14:43 by arjan.

Arjan Scherpenisse (1):

- Fixed mod_mailinglist compilation error on 0.7.x branch.

Release 0.7.4

Released on 2012-01-10 14:44 by arjan.

Arjan Scherpenisse (1):

- mod_twitter - use https for streaming API.

Marc Worrell (1):

- Fix for pivoting with a new resource.

Release 0.7.5

Released on 2012-03-11 09:04 by arjan.

Andreas Stenius (1):

- Fix dialog display issue when not using <html xmlns=...>

Arjan Scherpenisse (7):

- admin: Show error message when user tries to add the same page connection twice.
- LiveValidation: use the same e-mail regexp as in the backend.
- mod_import_wordpress: More feedback, convert text to proper paragraphs, support for 1.1 WXR schema.
- mod_twitter: Fixed converting unicode -> utf-8 in body text of received tweets.
- mod_oauth: Fixed access/request token uris to work with both GET and POST

Release 0.8.0

Welcome Zotonic 0.8.0, released on April 11, 2012. These are the changes for Zotonic release 0.8.0. The most important changes are summarized first, below that is a full "git shortlog" of all changes since release 0.7.

New core features

Module manager Module manager startup stability fixes, module dependencies and starting/stopping modules in the correct order.

Status site The site status got a redesign to be more in line with the current zotonic.com site. It now shows a friendly welcome message and requires a login to view / manage the running zotonic sites.

PostgreSQL We stabilized the pgsql connection pool in the presence of database connection failures and improvoved query timeout handling.

The "host" option in a site's config file is now optional. When not present it will be derived from the site's directory name.

New / updated modules

mod_oembed Provides an easier way to embed external content into your site, using the OEmbed standard.

mod_translation added support for RTL languages like Hebrew and Arabic in the admin interface. Content pages that are translated in multiple languages now have a separate URL for each language version. Translations of the admin interface were added for Irish, Spanish, Estonian and Polish.

mod_mailinglist Improved the mailinglist interface. It is now much easier to track to which list a page has been sent to, to preview the mailing and to view and manage bounced emails.

mod_development On Linux, development has been made easier by integrating inotify. Supports on-the-fly compilation of Erlang files, flushing caches, and compiling/minifying LESS/SCSS/Coffeescript.

Other minor features

New filter: `index_of`, which gives the index of an item in a list.

`filter_random:random/3` - create random sublist with length l.

`range` filter: easily generate lists with integers

Development process

The git master branch switched to using git submodules for the most important external dependencies.

Documentation got updated, most source files now have @doc tags which are generated and available online at from http://zotonic.com/documentation

Git shortlog

Alain O'Dea (2):

- Ignore compile output and runtime generated files
- Enhance PostgreSQL security

Andreas Stenius (27):

- z_pivot_rsc: Should also update pivot_category_nr when it is null.
- z_media_identify: Log identify errors.
- z_email_server: inc_timestamp/2 should increment by minutes.
- z_config: report config file parse errors, and terminate.
- site config: improved error reporting and host checks. (for issue #5)

- z_email_server: refactored the ?DELETE_AFTER define to be a customizable option.
- mod_backup: lookup db settings from pgsql_pool.
- m_edge: add set_sequence/4 to control edges.
- admin action "dialog_new_rsc": trigger custom actions for newly created resources.
- mod_base/filters/replace_args: new filter (#193).
- format filter: support filtering binaries. (#251)
- Windows: Update build.cmd for renamed webmachine -> webzmachine.
- mod_signup: urls are returned as binaries.
- Added Michael Connors for his Irish translations.
- Avoid // in path of found Makefiles.
- Pull in lager using git:// rather than https.
- Keep track of row number inside {# ... #}-comments.
- Fix dialog display issue when not using <html xmlns=...>
- fix for prev_year from feb 29.

Arjan Scherpenisse (245):

- Add 'no session' dispatch rule option to prevent starting a session on a page.
- Add check in zotonic-addsite to see if the given site name is valid.
- Add distinctive icon for internal link in TinyMCE. Fixes #189
- Add some cache blocks in case we get HN'd/slashdotted :-)
- Add z_context:{get,set}_cookie
- Added +x permission to zotonic-status script.
- Added .cw (Curaçao) to mod_l10n.
- Added .empty file for extensions dir
- Added 2 new translation template files.
- Added François Cardinaux to contributors, fix docstring.
- Added PUT and DELETE as accepted methods for API calls.
- Added a file with the translators per language.
- Added administration of worker processes to email server.
- Added application/x-font-woff for webfonts.
- Added blog section to www.zotonic.com
- Added commands to enable, disable sites
- Added date_start_year= and date_end_year= search filters to filter on year of date start/end.
- Added dutch month/day names to mod_l10n.
- Added dutch translation for mod_survey; small template tweaks. Fixes #205
- Added empty _language_attrs.tpl to admin module for when mod_translation is disabled.
- Added gen_smtp as a submodule.
- Added l10n_date:monthname_short/2 for short month names.
- Added lager as logging framework
- Added missing file

- Added mod_signal to the default list of installed modules.
- Added option email_bounce_override to override bounce email address.
- Added resource_menu_admin_menu again which was removed by accident.
- Added sass support to mod_development
- Added spanish translation and install the spanish language by default and enable it.
- Added support for scanning module .erl files for ?__ syntax.
- Added support for updating Zotonic and sites over Git in zotonic_status site.
- Added support for validation error message on radio elements.
- Added support for varying overview lists in the admin on category.
- Added the ability to use the resource_api handler for any URL.
- Added z_utils:percent_encode/1 function.
- Addressed the issues in the backup module. Fixes #220
- Admin link dialog: possibility to add a preconfigured list of defaults.
- Admin: remove tooltip from media attachment to fix dragging images to the right.
- Again fix the embedding of images in TinyMCE. Fixes #286
- Allow id to be either number or unique name for resource_admin_edit.
- Allow modules to override admin TinyMCE options which were originally set in admin-common.js
- Automatic make of changed .erl files works
- Bugfixes in m_edge:replace/4.
- Completely remove cufon from zotonic_status site, remove stats page
- Deal with spaces in provider name for embed template lookup.
- Disregard stderr output from identify command. Fixes issue #206
- Do not use sass caching
- Enable/disable now starts/stop the site on the node.
- Export z_pivot_rsc:insert_queue/2, for the delayed pivoting of a single rsc.
- Export z_session_manager:get_session_id/1.
- Facebook: Add possibility to redirect to a custom signup failure page.
- First work on module upgrader.
- Fix calls to z_sites_dispatcher:update_dispatchinfo/0
- Fix compilation error in z_toposort
- Fix custom server header for Zotonic with the new webzmachine.
- Fix infinite recursion in sub_month/3 filter.
- Fix stylesheet issues in new hierarchical editor.
- Fix warnings in m_rsc_update
- Fixed crash in inotify server of mod_development.
- d picture rotation detection by tweaking the parser of the output of “exif -m -t Rotation”.
- Fixed quality=xx parameter to {% image %}.
- Fixed some more admin translations
- Fixed typo in TinyMCE. See #286

- Fixed z_utils:tempfile() to respect OS environment variables.
- Fixes in twitter/facebook for changed z_dispatcher:url_for return value :-/
- Forgot to use catinclude after media item add in admin.
- Gave a fresh new look to zotonic_status, similar to Zotonic.com.
- Generalized group_title_firstchar filter into group_firstchar.
- Get the persistent id in template using {{ m.persistent.persistent_id }}
- Greatly improved the mailinglist feedback.
- Implemented new schema mechanism in all Zotonic modules.
- Let m_rsc:get_raw/2 return empty list instead of undefined when result is not found
- Let z_lib_include handle an empty request. Fixes #283
- Let zotonicwww site also use manage_schema/2.
- LiveValidation: use the same e-mail regexp as in the backend.
- Logoff controller now respects 'p' argument.
- Make bin/zotonic compatible with python 3.x
- Make the "change category" option better accessible
- Make windows users happy when redirecting stderr.
- Make z_form_submit_validated_do more stable using $.each()
- Makefile - use "find" to locate every Makefile we need, including those behind symlinks.
- Makefile now inits/updates git submodules if any.
- Media: classify `application/*` media files as "document".
- Move webmachine -> webzmachine in its own repository.
- Moved translation-tabs initialization into mod_translation.
- New filter: index_of, which gives the index of an item in a list.
- OAuth: fix request/acces token with POST
- OEmbed: even better error reporting, and show preview image when creating item.
- OEmbed: make the gen_server site-dependent; do not crash when getting invalid http request.
- OEmbed: when adding an oembed video, set the title if it's not set yet.
- On win32, mime type returned as application/octet for all files.
- Only show text direction controls in TinyMCE when mod_translation is enabled.
- Pass all filters into filter2arg function. Fix background removal for JPG images.
- Prettified the zotonic status commandline script
- Re-added the option of test-sending a mailinglist page to a single address.
- Refactored m_media:replace_file_mime_ok to not use a nested transaction in the insert case.
- Refactored the collecting of dispatch rules.
- Removed already_sent check from mod_mailinglist, which is not needed since the new interface.
- Removed gen_smtp in preparation of it being a submodule
- Removed m_identity:{get,set}_props which were unused and not working.
- Replaced TinyMCE with latest version. Fixed zmedia plugin.
- way mod_logging notifies the log pages; it now uses mod_signal for inter-page communication.

- Rsc pivot: fix case where sometimes pivot title would say 'undefined' and refused to update.
- Show error message when user tries to add the same edge twice.
- Show language selector on admin media page. Fixes #253
- Simplify manage_schema/2 module call allowing to return a #datamodel{}.
- er *after* sites manager so we can directly collect dispatch rules in dispatcher's init/1.
- Support ISO timestamps with time zone (currently ignored)
- Support for "extensions"; system-wide extra user-defined gen_servers.
- Support for default value in session get / get_persistent
- Tooltip Y position is now dependent on its height. Fixes issue #207
- and Facebook modules now also use #logon_ready_page observe pattern after successful logon.
- Updated the zotonic_install script
- Updated varnish config example to a more recent Varnish version
- Use catinclude in show_media filter for more versatility
- Use newer rebar script for iconv.
- When postgres exists normally, dont print info report.
- action_admin_dialog_edit_basics: custom action= argument(s)
- admin: use catinclude for _edit_media template, so it can be overridden.
- filter_index_of: Removed debug statements
- lower/upper filters now try to convert their argument to a list if it's not.
- m_rsc_update emptied the pivot_date_* fields when `date_` fields where not part of the update.
- mod_admin: Made the title of uploaded file optional.
- mod_admin: Press "enter" now saves the edit page.
- mod_backup: make sure we have an archive dir before archiving.
- mod_development - Removed unneeded ensure_server message and commented out trap_exit
- mod_development - flush cache on dispatch rule change.
- mod_development - remove debug statement, fix sass syntax
- mod_development.erl: When detecting new template, flush all cache to make sure it is found.
- mod_development: Added LESS css on-the-fly compilation.
- mod_development: when discovering new .tpl in site, flush its cache.
- mod_facebook: make 'scope' parameter configurable.
- mod_import_csv: Added import button to admin status page.
- mod_import_csv: Added more flexible date import and support for publication start/end.
- mod_import_wordpress tweaks
- mod_l10n: added ru.po, ru.country.po
- mod_logging - Fix log message formatting error.
- mod_mailinglist - added bounce handling dialog.
- inglist - attach documents to the mailing for each 'hasdocument' edge instead of 'document'.
- mod_mailinglist - fix include reference to mailing footer template.
- inglist: When sending to single address or to bounces, do not send to subscriber_of edges.

- mod_oauth: do not assume GET
- mod_oauth: fix API authorization check when using OAuth.
- mod_oauth: fix for R15B, changed http_uri:parse/1 return format.
- mod_oauth: fix for accessing public services when authorized
- mod_oauth: more refactoring; API services defined in site modules now also work.
- mod_search: Add creator_id and modifier_id to search query options.
- mod_search: improve the previous/next search function by allowing other dates to be paged on.
- mod_survey - show a counter column in front of every survey result in the editor.
- mod_survey - show questions in the right order
- mod_survey: Added a survey results edit page to the admin.
- mod_survey: Limit entry of "name" field to 32 chars.
- ey: Propagate qargs into the survey templates, make possible to add default values to survey
- mod_survey: made questions configurably required or not.
- mod_survey: normalize survey question names with z_string:to_name/1 instead of with to_slug/1
- mod_survey: quick hack to put email validation on a field if you name it 'email'.
- mod_twitter - use https for streaming API.
- mod_twitter – support for login using Twitter, similar to mod_facebook.
- mod_twitter: Fixed converting unicode -> utf-8 in body text of received tweets.
- mod_twitter: fix redirecting to ready_page by storing it in the session.
- mod_twitter: remove invalid {verbose, trace} option.
- mos_survey: fix chart export when answer name changed for yesno questions.
- oauth: Added allowed methods for access/request token uris.
- resource_api: do not start session when not needed.
- search_query: fix Erlang warning about exported variable.
- tiny_mce: Fixed the disappearing of inline images. Fixes issue #203.
- z_convert added ip_to_long/1 and long_to_ip/1.
- z_convert: fix timezone parsing for formats like 2011-10-06T14:44:00+0200
- z_convert:to_json/1 now also accepts floating point numbers.
- z_datamodel: do not try to resolve 'undefined' in valid_for check
- z_db:ensure_table – added primary_key attribute for custom primary keys
- z_filewatcher_inotify - Change timer:send_after to erlang:send_after
- z_html: do not escape/strip HTML when a property name ends in _html.
- z_html:escape_props/1 - Make selecting escape function more safe.
- z_module_manager: schema upgrades are allowed to return a #datamodel{} now as well.
- z_session:restart/1 can now take #context{} as argument.
- zotonic-start: cd to $ZOTONIC before doing make. Fixes #218

Atilla Erdodi (4):

- support for per property acl settings (note: you need to implement your own acl module. no example provided yet.)

- page model
- removed unnecessary info messages

François Cardinaux (1):

- New filter to escape JSON strings added to mod_base.

Konstantin Nikiforov (7):

- z_session, m_persistent: move SQL into model, cleanup ugly code
- added .gitignore
- z_search: added recursive concat for complex #search_sql{}
- m_persistent: fixed push()
- fixed SQL RETURNING behaviour on empty result.
- filter_random:random/3 - create random sublist with length l.
- mod_l10n: added ru.po, ru.country.po

Maas-Maarten Zeeman (39):

- Sometimes somebody (e.g. google) uses a smallcaps dtd
- Added a range filter to easily generate lists with integers. Syntax: 18|range:70 -> [18, 19, .., 70] or 2012|range:1900:'-1' -> [2012, 2011, .., 1900]
- Accidentally removed, generated new template
- Add facebook graph queries
- Added admin page for facebook module
- Added configuration option to increase the maximum number of concurrent connections
- Added facebook model for fql and graph queries
- Added if argument for optional caching. Issue #296
- Added spaceless block to strip whitespace between tags. {% spaceless %}...{% endspaceless %}
- Also refacted the sort event, again backward compatible
- Also refactored postback_notify. The refactor is backward compatible for postbacks, but not for notifications. The notification now also contains the trigger and target ids of the elements. This can lead to crashes in code which did not use records for notifications.
- Apply filter to a block. {% filter upper %}This will {% endfilter %}
- Backward compatible refactor for drag and drop events
- Backward compatible refactor. Moved submit and postback to records in order to make things more clear and remove the _TriggerId, _TargetId (or was it vice versa?) code. Todo are drag, drop, postback_notify and sort events.
- Change to get an object picture via the facebook graph api
- Changed filenames of translation templates
- Changed md5 hash for hmac and use base64url encoding so pickles are url friendly
- Configurable max memory for depcache
- Copied macros not needed anymore
- Couple of wrong renames
- Fix for a nasty loop caused by heart when things fail. Issue #212
- Fixed parsing of quoted attributes. They can contain newlines
- Fixes a race condition in which slots is called before the module is started

- Fixes an error when closing the dialog
- Fixes the edit button of acl_simple_roles. Fixes issue #208
- Format validator converts javascripts re's to pcre re's. Allows unicode re's #242
- Generated fresh pot files
- Made the acl_simple_role admin templates translatable
- Refactor, Introduced with_connection to handle direct pgsql queries easier
- Refactored facebook code. Now it can do graph posts too
- Refactored z_service so it works with modules and methods with underscores
- Removed experimental module
- Renamed translation template files from en.pot to modulename.pot
- Store z_notifier observers in ets instead of a dict
- Support for webm video
- iolist support for to_lower and to_upper
- quote_plus is now exported by mochiweb_util, removed copied code

Marc Worrell (136):

- Add 'action' to the #rsc_update fold nofitication, so we can distinguish an insert from an update.
- Add alternative urls to the head for translations of the current page.
- Add http:// before links starting with www.
- Added 'with' support to value expressions. Example: {{ a with a=3 }}
- Added dependencies to modules. Fixes issue 230.
- Added download link to media on page.tpl
- _existing_module/1 which checks if a module name can be found (and loaded) as a module. This without creating a new atom when the module does not exist.
- Added flattr button
- Added fold set_user_language, sets the context language to the pref_language of an user.
- Added freebsd to iconv rebar config. Thanks to Thomas Legg.
- Added id_exclude search term. With thanks to Michael Connors.
- Added is_even filter. Thanks to Michael Connors
- Added new menu/hierarchy sorter. In use for menu and category editors.
- Added remark about optional host configuration and module dependencies.
- Added sha1 as filter and javascript.
- Added some module dependencies. Changes default module dependencies to include module name with the provides and default [base] with the depends. Refers to issue #230
- Added support for posting z_notify javascript notifications directly to a delegate module. This calls the event/2 handler in the module (instead of the z_notifier observer).
- Added tinyMCE plugin for inline text direction editing: zbdo
- Added url rewrite on dispatch and generation. Now in use for automatically adding z_language argument to all paths.
- Added {continue, NewMessage} return format to z_notifier:first/2.
- Allow binaries for the header values. Fixes issue #257

- Allow included template to be a variable, forcing a runtime include. Fixes issue #256
- Allow non-atoms as language when setting the context language, ignore lists that aren't languages.
- Changed http into httpc for R15 compatibility.
- Changed the rsc_update_done notification to a #rsc_update_done notification record.
- Changing startup sequence.
- Check on return value of module activation transaction. Fixes issue #255.
- Fix for URLs with only a language code. Suppress language codes in URL when no languages enabled. Refers to issue #258.
- Fix for a problem where the admin overview crashed when no category was given.
- Fix for detaching m:f with pid when pid is not alive anymore.
- Fix for do_feedback on a single input element.
- Fix for loading beam files.
- Fix for loading modules on the fly.
- Fix for range requests
- Fix for saving surname prefix when signing up.
- Fix for the case where the exif orientation is an empty string.
- Fix for url encoding values >= 16. With thanks to Charles Won.
- Fix for when the to-be-submitted form disappeared during a feedback wait.
- Fixes #215. Hide the label when there is a value in the overlayed input.
- Fixes #225. Filters image tags with references for /admin/media/preview.
- Fixes for is_required handling of survey. Make created nullable for a smoother upgrade.
- Graceful Not Found in missing lib images.
- Make module startup handle nested upgrades.
- Make sure that text can be prepended/appended into an existing html element. Fixes #214
- Making modules more robust. More to follow.
- Missing argument for string:tokens/2
- Module manager: Only start modules when their dependencies are running.
- Monitor webmachine, in case webmachine goes down without telling z_logger.
- Moved old datamodel/0 to the manage_schema/2.
- No acl check for is_published and visible_for.
- Only allow existing and enabled languages as a language prefix. This fixes issue #258.
- REST API: Added support for jsonp callbacks
- Set min height of tree editor, so a drop on an empty menu is possible.
- Set the language of rendered e-mails to the preferred language of the recipient_id (if any)
- Stabilizing the database connection pool and sites in the presence of database connection failures. Refers to issue #269
- Support for rtl languages. (Arabic and Hebrew)
- Use bind instead of live - as we run into event bubble problems.
- admin: Fix for positioning dialogs that are higher than the window height.
- admin: Fix for unlinking images, result of action was not shown.

- dispatcher: Fix for creating URLs when a parameter has a regexp pattern.
- email: Added default values for relay options. Added option smtp_bounce_email_override to override the VERP for all sites.
- erlydtl: Added {% ifelse ... %} support. fixes #303
- i18n: Added Farsi (Persian) as right-to-left language.
- m_media: Fixed problem where uploading a file with a pre-defined category resulted in a duplicate category_id SQL error.
- m_rsc: Added 'is_import' and 'no_touch' options, this makes it possible to import data from other sites and keep the original created/modified dates.
- m_rsc: More efficient 'exists' check for rsc records.
- mod_import_wordpress: Renamed title of module to be more inline with other import modules.
- mod_menu: Remove invisible menu items when requesting menu_flat or menu_trail. Issue #291
- tinymce: Updated to 3.4.7
- to_list/1 shouldn't flatten a list, to_flatlist/1 should.
- websockets: Added support for hybi17 websocket protocol "13". Thanks to code from Cowboy by Loïc Hoguin
- z_datetime: Fix for next year on feb 29.
- z_db: Added automatic retry for deadlocked transactions.
- z_db: Added optional timeout argument for (most) database functions. Defaults to ?PGSQL_TIMEOUT. Fixes #301
- z_depcache: Make the process dict flush safe for proc_lib process dict vars.
- z_notifier: Allow programmes to send scripts to connected pages. Allow signals to be a simple atom, for programming simplicity.
- z_notifier: Documented z_notifier notifications. Translated tuple into records.

Michael Connors (5):

- Add Month, Day and Country translations in French and Irish
- Added Irish translation
- Added date validator to Zotonic.

Paul Guyot (1):

- z_db: Fix bug where ensure_table generated bad SQL when adding a column with a default value.

Piotr Meyer (10):

- Added Polish translation

Taavi Talvik (4):

- Added Estonian translation

Release 0.8.1

Released on 2012-08-11 16:36 by arjan.

Ahmed Al-Saadi (1):

- Removed websocket response header to conform to the latest proposed websocket protocol standard (see http://tools.ietf.org/html/rfc6455#section-4.2.2, December 2011)

Arjan Scherpenisse (20):

- Fixed the "make docs" command in the Makefile
- Fix 2 typos.
- mod_mailinglist - depend on mod_logging.
- mod_mailinglist - address recipients by name in welcome/confirm/goodbye messages.
- mod_mailinglist: new filter to add recipient details to links in mails.
- mod_backup: Prefix backup files with the site's name.
- Dynamically calculate the ebin dirs when compiling .erl, to ensure deps/ ebin dir existence.
- Fix wording of SEO webmaster tools configuration. Fixes #310.
- Merge remote-tracking branch 'origin/release-0.8.x' into release-0.8.x
- Added optional argument to show_media filter which specifies the used template for media items.
- z_email_server: Force quoted-printable encoding on HTML and Text parts in emails.
- Show e-mail addresses that have been entered in the survey in an easy copy-pastable dialog.
- mod_survey: Added easily-printable output.
- mod_survey: Sort the printable list on name_surname by default. Sort argument from the query string is also supported.
- mod_survey: Use lowercase when sorting printable list
- Backport fixes to Twitter filter to 0.8.
- Fix blog skel indentation. Fixes #329

Maas-Maarten Zeeman (4):

- The session id was not stored when a new session was created. Hopefully this fixes #327.
- Fix for mochiweb:to_html bug with singleton tags. Fixes issue #328
- Make language dependent URL rewriting configurable. Fixes #315
- Fix for logoff option. Fixes #382

Marc Worrell (8):

- Fixed a problem when editing a newly added menu. This was already fixed in the new-admin-design branch.
- Fixed typo in attr name ('check' instead of 'checked')
- Fix cache key of page. Issue #313
- Added travis-ci config from master.
- mod_survey: Added some css to display survey questions.
- mod_survey: Better likert display
- mod_survey: Better matching display
- mod_survey: Better longanswer display

Release 0.8.2

Released on 2012-10-30 11:48 by arjan.

Arjan Scherpenisse (3):

- Full text search: use the default postgres tsearch catalogs.
- Add new webzmachine commit for Firefox Websockets fix.
- mod_import_wordpress: Support schema 1.2; fix XMerl unicode behaviour.

Artur Wilniewczyc (1):

- fix multipart form data parsing: add new data at the end of previous data

Marc Worrell (8):

- Fix vulnerability in the random number and id generation. Fixes #369
- Fix mixup of tr/pl iso codes. With thanks to Kunthar.
- Allow searching on texts like 'id:1234' to find a specific id.
- Allow query_id to be a page with 'haspart' objects. Don't throw exceptions on mismatch of categories (this allows a query to be inserted before the referred categories are inserted)
- Only make queries with the 'is_query_live' flag a live query. Also some extra updates to m_rsc_update for later REST API work. Fixes #344
- Also merge the sort terms from 'query_id' includes.
- Add 'publication_before/after' search terms. Fixes issue #361
- Fix query for haspart of a collection.

Release 0.9.0

Welcome Zotonic 0.9.0, released on December 17, 2012. These notes list the most important changes for this new feature release.

New core features

The ability was added to aid and promote "mobile first" development using automatic user agent classification. Amongst others, the template selection mechanism is now based on the detected user agent class.

All base HTML has moved to use the Twitter Bootstrap CSS framework instead of Atatonic. This includes the admin interface (which got a new design) and the base templates for the skeleton sites. All Atatonic CSS files have been removed from Zotonic. Cufon is also no longer included.

While editing resources it is now possible for editors to add different "blocks" of content to the page. Blocks can be custom defined and have their own templates.

Modules can now implement their own URL dispatch mechanisms through `#dispatch{}` notifications. They are triggered when the regular dispatch rules do not match.

A new major mode for Emacs has been developed, called `zotonic-tpl-mode`. See documentation at zotonic.com for details on how to install it.

Zotonic's core routines which serve a greater purpose, like date and string manipulation and HTML sanitization, has been moved out of the main repository into a new subproject called z_stdlib.

New documentation

The documentation of everything Zotonic has been completely revamped. We now use Sphinx (http://sphinx-doc.org/) to generate HTML documentation. Other output forms (PDF, epub) are in the planning as well.

Source code for the documentation is in the main repository under doc/, so that it can be kept better in sync with new features in the code base.

The new documentation is up at http://zotonic.com/docs/

New tags/changed tags

{% javascript %}...{% endjavascript %} Adds javascript that will be run after jQuery has been initialized. In dynamic content it will run after the DOM has been updated with the template where the Javascript was defined.

{% image %} new attribute: mediaclass. Image classes can be defined using property files in the template directory. Which image class definition is chosen depends on the user agent classification.

New and updated modules

mod_geomap New module: Provides mapping and geocoding.

mod_comment Added the possibility to have comments be moderated before submitting.

mod_survey mod_survey has been largely rewritten. Now uses the new 'blocks' structure for adding questions to the survey. Many new question types have been added, like file uploads and category selection. Also supports mailing the survey results and custom handlers.

mod_ssl New module: adds SSL support to sites. Previously only a single certificate could be used per Zotonic server. With this module each site can have its own HTTPS listeners and certificates. When you don't supply a certificate then a self-signed certificate and private key will be generated using the openssl utility.

mod_backup Adds basic revision control for editing resources in the admin interface. Every time a resource is saved, a backup version is made. Using a GUI you can inspect differences between versions and perform a rollback.

mod_rest A new module to interact with Zotonic's data model using a RESTful interface.

New filters

menu_subtree: Finds the menu below a particular resource id. Usage: m.rsc.main_menu.menu|menu_subtree:id

escape_link: Escapes a text, inserts <br/> tags, and makes anchor elements of all links in the text. The anchor elements have the 'nofollow' attribute.

sort: Sorts a list of resource ids based on their properties.

New custom tags

geomap_static Makes the HTML for a static map of a location. Uses the OpenStreetMaps tileset. Example usage:

{% geomap_static id=id n=3 %}

Show the location of the resource 'id' in a grid of 3x3 tiles.

Translations

Translations were added and updated for Dutch, German, Polish, Turkish.

Contributors

The following people were involved in this release:

Ahmed Al-Saadi, Alain O'Dea, Andreas Stenius, Arjan Scherpenisse, Artur Wilniewczyc, Barco You, François Cardinaux, Grzegorz Junka, Hans-Christian Espérer, Ivette Mrova, Joost Faber, Kunthar Daman, Maas-Maarten Zeeman, Marc Worrell, Motiejus Jakštys, Piotr Meyer and Tom Bradley.

Release 0.9.1

Released on 2013-03-14 19:31 by arjan.

Andreas Stenius (21):

- core: add support for 'optional include' templates.
- doc: Add *build:* tag to list of prefixes.
- doc: Add *translations:* tag to list of prefixes.
- doc: add links to other versions of the docs.
- doc: fix version links.
- doc: removed a tag from the version being browsed.
- doc: rename *translations:* to *translation:*.
- doc: rename link to 0.9
- doc: renamed sidebar template to be more generic.
- doc: update frontend-growl cookbook entry.
- mod_admin: optionally include country options.
- zotonic-tpl-mode: add testsuite.
- zotonic-tpl-mode: bind C-M-to zotonic-tpl-indent-buffer.
- zotonic-tpl-mode: do not treat underscore as being part of a word.
- zotonic-tpl-mode: fix for indenting consequtive closing template tags.
- zotonic-tpl-mode: fix html tag highlighting when attr value contains //.
- zotonic-tpl-mode: indentation fixes.

Arjan Scherpenisse (49):

- Ensure iconv is started
- New z_stdlib module
- basesite skel: Add show_media filter to home template
- build: Explicitly build lager first
- controller_api: Fix throwing a 404 error when service module not is found
- core: Fix use of hostname in startup / management scripts
- core: Include the request hostname in a dispatch rewrite notification
- core: Keep file extension when it is allowed for the file's mime type
- doc: Add paragraph about multiple assignments to *with* tag docs
- doc: Add some documentation about startup environment variables
- doc: Document how to override TinyMCE options
- doc: Document mod_tkvstore
- doc: Document the "remove" action
- doc: Document the arguments of the "lazy" scomp
- doc: Explain API service naming in more detail
- doc: Explain the make_list filter to force model evaluation
- doc: Improve wording on replace filter
- doc: show_media filter: correct the default value for media dimensions

- mod_admin: Add button to run schema install from the site's module again
- mod_admin: Add inlinepopups TinyMCE plugin.
- mod_admin: Fix redirect to admin page when using admin logon form
- mod_admin: Fix some errors in new rsc dialog
- mod_admin: Fix view button class on readonly edit page
- mod_admin: Prevent id confict between forms
- mod_admin: clicking the "move" indicator of an edge should not open the details dialog
- mod_admin_config: Truncate large config values in the config table
- mod_admin_identity: Show inactive users in a lightgrey color
- mod_admin_predicate: Fix escaping of translated text in predicate help popup
- mod_base: Add z_stream_restart() function to zotonic-1.0.js
- mod_base: CSS typo
- mod_base: Fix 'width' argument for Bootstrap modal
- mod_base: Fix remember password for Chrome
- mod_base: Fix rendering issue in TinyMCE when scrolling
- mod_base: Log error in widgetmanager when evaluating the JSON data attribute fails
- mod_base: Remove accidental console.log
- mod_base: make controller_page:get_id/1 return undefined when no id is found
- mod_signup: Add fold notification for determining which fields to validate
- mod_signup: Allow signups with email only
- mod_survey: New question type "multiple choice"
- mod_translation: Add zh to languages list, add Shuyu Wang to translators
- mod_translation: Make checkbox for setting language as part of the URL
- mod_twitter: Change URL where one can resolve a Twitter ID
- scripts: Fix typo in zotonic-update
- scripts: When zotonic is not running, echo the output of erl_call
- translation: Add complete Chinese(zh) translation
- z_media_identify: Fix typo in re-run command on identify fail

Feather Andelf (1):

- translations: Chinese(zh) translations by Shuyu Wang aka andelf

Grzegorz Junka (2):

- Update controller_static_pages documentation
- skel: Update default nodb skeleton config

Ilyas Gasanov (1):

- translation: Added more complete Russian translations

Maas-Maarten Zeeman (1):

- Check if ua_classifier is functioning at startup.

Marc Worrell (57):

- core: add option 'use_absolute_url' to args and context of url generation (pages and media)

- core: added 'runtime' option to 'include' tag. (Renamed from undocumented and legacy 'scomp' argument.)
- core: added option to force the protocol of user facing urls. This is useful when a proxy translates between https and http.
- core: added z:ld(modulename), z:restart(sitename) and z:flush(sitename)
- core: added z_render:update_iframe/3. Used to render a template and place it into an iframe.
- core: allow an included file to use extends/overrules within the scope of the included file.
- core: allow anonymous access to the page_url_abs property.
- core: better error messages for z_db:q and q1 in case of sql errors.
- core: cache result of check if a specific module is enabled.
- core: ensure resource ids passed to the ACL are integers. This makes ACL modules simpler.
- core: fix for forced host/protocol redirects by mod_ssl
- core: fix return value of protocol redirect.
- core: fix spec of function.
- core: m_rsc - sanitize the uri input
- core: merges z_user_agent modifications from master (e3a6605ab3f9c48bda7eaa2d37c12bc7ad58a67b)
- core: more error messages instead of match error.
- core: prevent sites dispatcher from crashing when redirect page dispatch rule is missing.
- core: psql: better error message for insert/delete
- core: remove start/stop of iconv (is now eiconv)
- core: removed some ununsed var warnings.
- core: return 'undefined' for m_rsc:rid([]) lookups.
- core: show a stack trace when a periodic task crashes.
- core: use z_context:abs_url/1 when redirecting to the main site url. Also make it a permanent redirect (301)
- deps: add eiconv as git submodule.
- doc: add google analytics code.
- doc: added some missing smtp options
- doc: fix in docs for split_in and vsplit_in filters.
- eiconv: fix for warning on linux about missing iconv library.
- facebook/twitter: fix redirects, page_url is now a binary.
- mod_admin_identity: added #identity_verified{} notitication. Used to mark a verified identity to an user.
- mod_admin_identity: added a control to add/verify e-mail identities instead of the e-mail address input for a person.
- mod_admin_identity: fix for update_done notifications other than insert and update.
- mod_authentication: fixes for password reset e-mail and some error feedback messages.
- mod_base/core: added 'qarg' argument to postback actions. Enables adding the value of input elements as query args to the postback.
- mod_base: don't handle onsubmit validations for 'nosubmit' input elements.
- mod_base: moved the e-mail validation regexp to z_stdlib's z_email_utils.erl.
- mod_base: remove all zmodal masks when removing existing zmodals.

- mod_base_site: add style for meta share button
- mod_export: added generic export controller and notifications. (docs will be added)
- mod_l10n: added translations for relative time desciptions in the core z_datetime.erl.
- mod_search: allow nested category lists, allowing for more flexible category specification.
- mod_seo: fix for showing the summary when there is no seo description for the page.
- mod_seo: let search engines only index pages with real translation.
- mod_seo_sitemap: added category specific setting for update frequency and indexing priority.
- mod_seo_sitemap: fix for url generation, should page_url_abs.
- mod_seo_sitemap: generate language version links when mod_translation is enabled.
- mod_seo_sitemap: removed extra template, now uses catinclude.
- mod_seo_sitemap: use 'all catinclude' to fetch the extra seo options.
- mod_survey: added 'answers' option to survey_start, allowing answers to be preset (or added)
- mod_survey: fix for yes/no and true/false questions with immediate submit.
- modules: check if table is present before trying to create it.
- smtp: fix edocs parsing problem on binary notation in comments.
- smtp: use eiconv instead of iconv. Add better bounce recognition and is_bulk/is_auto flags for received e-mail.

Mgpld (1):

- mod_base: Modal html title in z.dialog

furiston (1):

- translations: Turkish translations by Evren Kutar aka furiston

Release 0.9.2

Released on 2013-07-02 20:53 by arjan.

Andreas Stenius (7):

- zotonic-tpl-mode: optimize font-lock regexp.
- zotonic-tpl-mode: optimize font-lock regexp #2.
- zotonic-tpl-mode: refactor tag soup indent calculations.
- zotonic-tpl-mode: autocomplete closing tag soup tags.
- zotonic-tpl-mode: improved auto close tag.
- zotonic-tpl-mode: fix bug in find tag soup open tag when encountering a self closed tag.
- doc: minor tweaks to the dispatch documentation.

Arjan Scherpenisse (19):

- mod_logging: Fix live display of log messages
- mod_base: Fix filter_capfirst to support i18n strings
- core: Add 'default_colorspace' global option to set the default ImageMagick colorspace
- mod_import_csv: Correctly treat .csv files as UTF-8
- Fix z_sites_dispatcher:split_host/2 for 'none' value
- zotonic_status: Add API for global site status check

- doc: Update preinstall instructions for Ubuntu
- mod_mailinglist: New dialog which lets you combine lists
- mod_base: Fix controller_static_pages's rendering of template files
- mod_mailinglist: Make bounce dialog nicer
- mod_mailinglist: Only show addresses in bounce dialog for the current mailing page
- mod_backup: Also prefix the periodic backups with the site name
- mod_admin: Resources in the meta category can have 'features'
- core: Fix startup script distributed detection
- core: eiconv should do a make clean, not a rebar clean
- core: zotonic CLI command now refers to website for help on subcommands
- code: Add m_edge.id[123].predicate[456] syntax to look up an edge id from a triple
- mod_acl_simple_roles: Fix bug where visible_for would be reset when it was not part of a rsc update call.
- mod_admin_identity: Add option to send username/password to the user over e-mail

Bryan Stenson (1):

- fix typo in default data

Marc Worrell (67):

- core: show info messages when a site starts/stops.
- core: erlydtl - allow lookup in a list of n-tuples (n>2) by first tuple element.
- core: i18n - don't replace assume an empty translation should be added as the original lookup string. This fixes an issue where an undefined translation overwrites an existing translation from another module.
- core: remove accidently added notes about federated zotonic.
- core: i18n - removed warning
- docs: fix build on OS X (BSD sed is a bit different than on Linux)
- docs: simpler echo to file, remove temp file.
- docs: document #foo.id syntax
- core: added support for signing up with facebook for a known user. Moved some check functions to m_identity. Fixed problem where an email identity could be added multiple times (when signing up using FB)
- Fix for close button on persistent notices.
- mod_signup: fix spec
- mod_signup: fix typespec
- mod_facebook: fixes to redirects etc.
- mod_base: added 'only_text' option to alert. Corrected documentation.
- mod_authentication: remove reload of other open 'tabs' on log on/off. This gives a race condition in Chrome as Chrome doesn't close the Websockets channel during redirects (which are setting the new cookies)
- core: added z:log_level(debug | info | warning | error) to set the console backend log level.
- mod_development: show debug message when reloading a module.
- mod_base: cherry pick from 5d252cb. Issue #491
- mod_base: remove part of merge.
- mod_l10n: japanese translation for days and months.

- mod_base_site: remove scrollbars from share dialog
- core: don't crash when an identity has duplicates.
- mod_base_site: separate copyright template. Use tablet/desktop footer for (smart)phone.
- mochiweb: merged fix for missing summer time hour.
- mod_oembed: fixes for provider list. Added oembed_client:providers/2 call for debugging. Added lager logging.
- mod_import_csv: connected reset checkbox, now also forces re-import of items. Added support for 'medium_url' column name, downloads and imports an url for a medium item. Added lager logging. Added filtering of escaped quotes.
- core: added extra arg to m_media:replace/insert for m_rsc_update options.
- core: fix for upload from url.
- mod_base: simpler and better z.clickable.js
- mod_custom_redirect: merge of 0f8d234 from master
- mod_custom_redirect: fix delete buttons, and type of submit button
- mod_custom_redirect: check for access permission when saving configuration.
- core: fix for mimetypes where 'image/jpeg' returns preferred extension '.jpe' (instead of '.jpg')
- doc: added documentation about the site.bounce_email_override config.
- Fix for checking delivery status - header decode was using wrong function.
- core: smtp: use os:timestamp() instead of now(). Added guards and more strict tests for inc_timestamp/sent handling
- core: the hostname can be 'null', handle it as 'undefined'
- core: make more robust against illegal page paths from hacking attempts.
- mod_l10n: Added South Sudan. Needs additional translations
- mod_authentication: make the password reset available when logged on
- mod_authentication: fixed reminder link in error message.
- mod_base: fixes for wired id css selector support in combination with a postback. This hack should be replaced with proper 'selector' support for wired targets.
- core: don't change the language on internal page_path rewrites. Fixes #559
- smtp: fix e-mail receive.
- deps: new z_stdlib.
- mod_admin/mod_admin_frontend: made admin more responsive. Added first version of frontend editor based on mod_admin.
- mod_admin: fix '&' in doc description as it crashed the doc generation.
- core: fix failed merge.
- mod_base: copy if_undefined filter from master.
- mod_admin: fix body top padding for smaller screens.
- mod_admin: Make the modal popup in the admin_frontend as wide as in the admin.
- mod_menu: use short title; absolute positioning of item buttons. mod_admin_frontend: minor textual changes, add dutch translation
- mod_menu: show warning when a resource is unpublished.
- mod_admin/mod_menu: added option so pass tab to connect/new dialog.

- mod_admin_frontend: set default connect tab to 'new'
- mod_admin_frontend: preselect the 'text' category for page linking.
- mod_admin: make the logon more responsive (header row-fluid; some styling)
- mod_admin_frontend: fixes for editing blocks and language selections.
- mod_admin: nicer fixed save button bar
- mod_admin_frontend: see images in the tinymce. Nicer save/cancel bar
- mod_menu: small textual change
- mod_admin: fix tinymce image preview.
- mod_admin_frontend: remove debug javascript.
- mod_base/mod_translation: add filter 'filter'; added m.translation model for easy language access.
- mod_ssl: change behaviour of 'is_secure' setting; redirect to https when not specified otherwise.
- mod_ssl: added an 'is_ssl' configuration to force a site to be served over ssl. Use in combination with 'is_secure'.
- Merge pull request #579 from fenek/install-timeout-fix

Release 0.9.3

Released on 2013-09-21 09:37 by arjan.

The main focus of this release is to make it compatible with the Erlang R16 series. Other changes are below, grouped by developer.

Arjan Scherpenisse (24):

- core: Fix .ogg/.ogv to be correctly identified as audio/video.
- core: Implement "center of gravity" option for crops
- doc: Fix syntax error in generated sphinx conf.py file
- doc: Move the section on git commit messages to the "contributing source code" section.
- mod_acl_simple_roles: Fix typo
- mod_admin: Create interface to pick the cropping center on images.
- mod_admin: Do foldr over the blocks instead of foldl
- mod_admin: Speed up admin edit page JS performance
- mod_admin_identity: Also offer the "send welcome" option when changing the username / password
- mod_admin_predicate: Fix category tree for fluid layout change
- mod_base: Add optional tabindex to button tag.
- mod_development: Fix crashing file watcher process while editing templates
- mod_oembed: Better error reporting and "fix all" button
- mod_oembed: Fix setting media title when programmatically importing an oembed media item.
- mod_oembed: Prevent crash in media tag when no oembed metadata is found.
- mod_survey: Fix crashing survey results
- mod_survey: Improvements (placeholder texts for input fields, translateable validation msgs)
- mod_twitter: Fix Twitter logon and API access for changed 1.1 API

Marc Worrell (21):

- core: added 'magick' image filter options. Fixed mime lookup if lossless filter was inside a mediaclass definition.
- core: added z:ld/0 to load all re-compiled beam files.
- core: filenames might have a space, escape it befor os:cmd/1
- core: fixed mediaclass lookup - now all lookups are done as documented with adherence to the module priority and the ua lookup.
- core: move rsc property access check inside p/3 instead of the model access.
- core: remove debug statement from z_acl.
- core: removed debug messages from z_media_tag.
- deps: newest z_stdlib.
- m_identity: ensure that non-unique identities have is_unique set to 'undefined' (instead of false)
- m_media: accept upload of 'data:' urls for replace_url and insert_url functions.
- mod_admin: fix replacing media items.
- mod_authentication: send e-mails to all matching accounts, if and only if a username is set.
- mod_authenticaton: fix input sizes for password, and e-mail/username fields.
- mod_base: added 'is_result_render' option to 'moreresults' action. Allows render of all results at once instead of per result item.
- mod_base: added touch punch to enable drag/drop and other touch events on mobile touch devices. Fixes #597
- mod_base: fix hybi00 websocket connections. Fixes #637
- mod_base: fix in LiveValidation for resetting error class on textareas. Fixes #601
- mod_base_site: add class 'wrapper' for content, allows for better styling. Hide breadcrumb and subnav when the current page is the only page in the breadcrumb or subnav.
- mod_geomap: Perform country lookups by iso code - works better with OpenStreetMaps API.
- mod_geomap: export lookup functions, add hard coded lookups for exceptions.
- mod_geomap: load cluster popups via template. Disable google geo lookups for half an hour if quota exceeded. Cleanup of some js code. Perform geo lookups in same process as pivot, prevents too many parallel lookups when re-pivoting.

Release 0.9.5

Released on 2014-04-18 16:55 by arjan.

Arjan Scherpenisse (20):

- mod_admin: Remove warning in m_admin_blocks
- doc: Document how to set m_config values in the site's config
- mod_mailinglist: Fix button order of delete confirmation dialog.
- mod_base: Match the websocket protocol on the WS handshake.
- z_stdlib: new submodule
- New eiconv submodule
- mod_survey: in printable overview, use fallback to column name when no column caption has been set.
- mod_survey: Add m_survey:get_questions/2 function
- mod_base: controller_page - check if page path equals the request path

- mod_base: controller_page - make canonical redirect behavior configurable.
- doc: Clarify the doc page about custom pivots
- core: basesite skeleton: install mod_acl_adminonly before mod_authentication
- core: z_utils:json_escape now points to right function in z_json
- core: Update dispatch info on start and stop of a site.
- core: Show error message in admin when a module fails to start
- doc: Update mod_twitter to show how to do Twitter login
- doc: Explain what a *query resource* is
- doc: Explain all default fields in a resource
- mod_mailinglist: Do not crash when recipientdetails filter encounters an empty body

Cillian de Róiste (1):

- doc: Add cookbook item on creating custom content blocks

Grzegorz Junka (1):

- core: Fix in Zotonic after removing access to the webmachine_request:send_response/2 function

Maas-Maarten Zeeman (4):

- core: Make sure js boolean values are converted correctly.
- mod_base: Make sure no messages are left in the request process's mailbox. Fixes #676
- mod_base: Limit the number of websocket reconnects. Fixes #675
- core: Prevent to detach comet controllers from page sessions which are already stopped.

Marc Worrell (19):

- core: periodic process to unlink deleted medium files (and their generated previews)
- z_stdlib: fixes for escape/escape_check of proplists.
- deps: new z_stdlib
- mod_import_csv: improved parsing of CSV files, now handles quoted fields and performs automatic translation from latin-1 to utf-8
- mod_import_csv: remove stray underscore character from module name...
- mod_import_csv: remove superfluous lager messages about escaped chars.
- mod_base: reapply change from d5604eca1f83491c37adf7981ff1cc598b7c57e2
- Fix on fix. start_timer returns a Ref, not {ok, Ref}.
- mod_base: 0.9.x doesn't know about z_ua.
- core: use lower case extensions when guessing mime or file extension.
- core: for exif lookups, add LANG=en for sensible exif program output (not for win32 as shell var is used))
- mod_base: use window.location.host for websocket stream, unless configured otherwise. Also add try/catch around websocket connect, to catch potential IE10 security exception.
- core: new state machine for maintaining websocket/comet push connections. Perpetual comet loop which only polls if there is no active websocket connection. Websocket connection checks with a periodic ping/pong if the connection works.
- core: fix difference between master and 0.9 regarding z_stream_start.
- mod_base: fix for subdomain polling loop.
- New gen_smtp from zotonic/gen_smtp/master

- mod_admin_identity: fix identity insert race condition by moving email identity check into the #rsc_update event instead of the async #rsc_update_done event.
- base: fix NS_ERROR_XPC_BAD_CONVERT_JS on Firefox. Fixes #718
- core: fix SVG mime type mapping from ImageMagick identify.

Release 0.10.0

Welcome Zotonic 0.10.0, released on April 18, 2014. These notes list the most important changes for this new feature release.

New core features

The minimum required Erlang version is now release R15B03. Zotonic is always supporting up to major 2 versions behind the current stable Erlang release.

We now use the de-factor Erlang package manager rebar for the management of the subprojects.

Websocket / comet (re)connection handling have been greatly improved. A state machine now manages the process on the server and keeps the connection running also in the long run.

The Google Summer of Code-sponsored student Mawuli Adzaku provided excellent work on the Zotonic CLI for management (search, installation, ...) of external Zotonic modules such as those from http://modules.zotonic.com/

The media handling subsystem was heavily modified in this release. It is now much easier to add custom hooks to do custom media processing when a media file has been uploaded. The following notifications were added as extension points:

- `#media_identify_extension{}` to make custom mappings from a mime-type to an extension
- `#media_identify_file{}` to override media file identification
- `#media_update_done{}` when the media metadata has been stored in the database

For SEO purposes, a new table *rsc_page_path_log* was added, that tracks old page_path names from *resources* so that old urls keep working when the page path changes. Old URLs will be redirected to the new location instead.

Handling legacy username/password authentication can now be done by leveraging the new `#identity_password_match{}` notification.

New and updated modules

mod_seo The two modules mod_seo and mod_seo_google have been merged into a single module. Support for Bing webmaster verification has been added.

mod_artwork This module now ships with the font-awesome icon set.

mod_admin A much requested feature to set the "center of gravity" for images has been added in the admin. The 'crop' filter takes this center into account.

mod_admin_frontend This new module will provide admin-alike editing of resources, menu-trees and collections for non-admin users in the frontend of the website, by the re-use of admin templates in the frontend site. This module was backported to the 0.9 series.

mod_admin_stats The admin stats module adds a new admin page presenting a live view of system statistics for things like number of requests, request processing times, database and cache queries etc.

mod_custom_redirect This module provides hooks to redirect unknown request paths or hosts to other locations. It adds a page to the admin to define and maintian these redirects. This module was backported to the 0.9 series.

mod_video A new module for video conversion and playback. It uses theffmpeg tool to convert uploaded movies to h264 and generate a poster image. Included are default templates for movie playback.

mod_mqtt Interfaces to MQTT publish/subscribe topics. Adds publish/subscribe between the browser, server and modules. Includes access control and topic mapping for external MQTT applications.

Template system improvements

The template include system was made more powerful by adding options to force the runtime include of a template and to make the include optional (e.g. not crashing when a template is not found).

The option was added (to *mod_development* (page 289)) to generate HTML comments with the paths to the included files embedded for easy debugging.

Besides this, the erlydtl_runtime can now directly lookup values in mochiweb JSON structures, making template syntax more straighforward.

New template filters

trim Removes whitespace at the start and end of a string.

pickle Pickle an Erlang value so that it can be safely submitted with a form.

tokens Returns a list of tokens from input string, separated by the characters in the filter argument.

filter Filters a list on the value of a property.

sort Sorts lists in various ways

if_undefined Tests whether a value is undefined, returning the given argument.

menu_is_visible Filters a list of menu items on visibility.

menu_rsc Get a "flat" of menu parents

Translations

Translations were added and updated for Chinese (ZH), Russian, Turkish, Dutch, Polish, Portugese, French and German.

Contributors

The following people were involved in this release:

Alexander Stein, Andreas Stenius, Arjan Scherpenisse, Arthur Clemens, Bryan Stenson, Carlo Pires, Cillian de Róiste, Feather Andelf, furiston, Grzegorz Junka, Ilyas Gasanov, Jarimatti Valkonen, Maas-Maarten Zeeman, Marc Worrell, Mawuli Adzaku, Mgpld, Piotr Nosek, Simon Smithies and Steffen Hanikel.

Release 0.10.1

Released on 2014-06-17 20:27 by arjan.

Release highlights

New core features

The properties of the resource model (m_rsc) can now be easily be inspected with the 'print tag' and iterated over.

New core features

The properties of the resource model (m_rsc) can now be easily be inspected with the 'print tag' and iterated over.

Updated modules

mod_development Added dispatch debugging and explanation. Added checkbox to disable the api-service / `api/development/recompile`

mod_admin_modules Add "configure" button in the module manager for modules which contain a template called `_admin_configure_module.tpl`.

New notification

Added the notification `request_context`. This is a foldl with the *Context* and is called after the request's query arguments are parsed using `z_context:ensure_qs/1`. It can be used to perform transformations or actions based on the query arguments.

Commits overview

Arjan Scherpenisse (11):

- build: Always use a downloaded rebar
- core: Add pivot columns for location lat / lng.
- core: Add zotonic_dispatch_path to z_context blacklist
- core: Fix comet/WS when site streamhost is undefined and redirect = false
- core: Let Travis build 0.10.x
- core: recode iso639 module to UTF-8
- doc: Clarify why Zotonic cannot directly use port 80
- mod_admin: Add option 'show_date' to 'latest resources' admin dashboard widget
- mod_admin: Correct vertical centering of connection dialog
- mod_admin_modules: Add "configure" button on module manager
- mod_base: Vertically center the z_dialog when it is opened

Arthur Clemens (19):

- doc: Add OS support, change name to Requirements
- doc: Improve docs sidebar layout a bit
- doc: Rename 'tutorials' to 'installation'
- doc: Tuning docs on services and mod_oauth
- doc: add example to filter on depiction
- doc: add link to manual dispatch
- doc: document restart command
- doc: elaborate conditional display of widget
- doc: fix 'button title' to 'button text'
- doc: show_media tag TLC
- doc: tweak highlight and see also colors

- doc: update Zotonic shell
- make new page button pass current category
- mod_admin: disable reindex button while indexing
- mod_admin: improve feedback
- mod_admin: live update of pivot queue count
- mod_editor_tinymce: introduce z_editor

Marc Worrell (24):

- core: added 'request_context' notification. Triggered after a request's query arguments are parsed.
- core: always set the user explicitly in the session. Don't copy the user from the current context.
- core: bump version to 0.10.0p1
- core: fix problem where z_db:insert can hang when running out of db connections.
- core: let the template {% print m.rsc[1] %} print all rescource properties. Same for {% for k,v in m.rsc[1] %}.%
- core: more binary/list changes (due to z_string changes)
- core: set default schema for sites without dbschema config.
- core: set the option for Erlang 17.0 and later. Issue #764
- core: set the zotonic_dispatch_path to the path before an internal resource redirec.
- docs: add documentation for zotonic_dispatch_path_rewrite var
- docs: add more mod_development documentation.
- docs: add note about 0.10.0p1
- docs: added preliminary 0.10.1 release notes.
- mod_admin_frontend: fix for introduction of z_editor. Also sync the language tabs.
- mod_development: add access control to the dispatch rule debugger.
- mod_development: add checkbox to enable or disable the development/recompile api.
- mod_development: add human-readable form for rewrite_nomatch. Always show the final dispatch.
- mod_development: added request dispatching debug. This shows the steps of the dispatcher in matching a dispatch rule, including all bindings, rewrites etc.
- mod_development: added template lookup tool to check the template selection. Optionally add a resource castegory. Shows the selected template per user-agent class.
- mod_development: fix build of edoc.
- mod_development: fix problem where the dispatch-debugging interfered with ssl connection redirects.
- mod_development: show final dispatch after #dispatch_rewrite.
- mod_oembed: fix for z_string:to_name/1 now returning a binary.
- mod_survey: fix display of likert. Values were reversed from real values.

Mawuli Adzaku (1):

- mod_base: Add 'md5' filter to translate a string to an md5 hex value

Release 0.10.2

Released on 2014-10-01 20:16 by arjan.

Arjan Scherpenisse (13):

- mod_menu: Add menuexport JSON service
- mod_survey: Return the (possibly generated) submission id, after insert
- mod_admin: Put the unicode quotes of the link message in binaries
- core: Flush z_datamodel after managing new categories
- mod_twitter: Fix twitter OAuth logon
- mod_base: Export controller_logon:set_rememberme_cookie/2
- doc: clarify using categories defined by other modules in manage_schema
- core: Always normalize rsc blocks
- mod_survey: Align ACL check of printable results page
- core: Fix name-to-id cache invalidation
- mod_admin: Add option to create outgoing edges in new page dialog.
- mod_survey: Fix dispatch rule ACL access
- mod_oauth: No ACL check on authorize URL; just check if logged in

Arthur Clemens (19):

- minor translation and typo fixes
- docs: small improvements to link and unlink action
- doc: mention named wire actions in wire doc
- Remove redundant is_protected check on resource
- doc: update menu documentation to reflect actual output of the module
- mod_menu: add param class to menu
- doc: add reference to to_integer
- core: increase maximum image size for previews
- doc: describe attributes removebg and upscale
- doc: document md5 filter
- doc: document filter default
- doc: fix module install command
- translation: typo
- Fix various NL translation inconsistencies
- Fix NL translation
- Only use pre block if error_dump is available
- Add clause for 404
- mod_acl_simple_roles: add module names to titles
- mod_oauth: change required access to mod_oauth

David de Boer (3):

- Fix edges not being added in manage_schema
- Allow to set is_protected from manage_schema

- mod_admin: Fix typo

Marc Worrell (12):

- mod_video_embed: correctly fetch preview images for youtube videos with a '#' in the url.
- mod_video: fixes for handling binaries returned from the z_string and other function (used to be lists)
- Merge branch 'release-0.10.x' of github.com:zotonic/zotonic into release-0.10.x
- core: minor fix in mime-lookup for files not recognized by 'identify'
- core: allow binary results from validations. For now map them to lists, later with the binary webmachine replace this with binary only.
- core: fix problem where rsc_gone insertion would crash if there is already a 'gone' entry for the rsc. This is in the category 'should not happen', but also happened in production.
- core: add infinite timeout to dispatcher calls, prevent crashes on busy or swapping machines.
- core: add DB_PROPS macro to ease transition from 0.10 to 0.11
- Merge pull request #799 from driebit/is-protected-schema
- mod_survey: better survey results display.
- core: fixes for tinymce initialization and language tab sync in the admin
- mod_admin_identity: let's stop sending the password in clear text.

Release 0.11.0

Welcome Zotonic 0.11.0, released on October 8, 2014. These notes list the most important changes for this new feature release.

Note: For upgrading to this release from an older Zotonic release, please read the *Upgrade notes* (page 190) carefully, as some backward-incompatible changes have been made.

Full timezone support

Timezone support was added to the core. All dates are now stored in UTC. Existing resources with old dates (in local time!) are converted on read, assuming the configured server timezone. You need to set the timezone in the `zotonic.config` file, for example:

```
{timezone, "Europe/Berlin"}
```

A site-specific timezone can de set with the `mod_l10n.timezone` configuration.

Database driver and pool

The database driver and pool has been replaced by the standard `epgsql` and `poolboy` Erlang applications. This means we no longer use a special forked Zotonic version of the epgsql Postgres driver. As a nice side effect, database queries run 5-10% faster on the new database code. See the test report here: https://gist.github.com/arjan/70e5ec0ecaf98b19a348

Global changes

The default place for user-defined sites and external modules has been changed to the defaults `user/sites` and `user/modules`, respectively.

Also, the global file `priv/config` has been obsoleted in place of a new global configuration file, `~/.zotonic/zotonic.config`. Zotonic actually looks in several places for its global configuration file, including `/etc/zotonic/zotonic.config`. See *Global configuration* (page 490) for all information on this topic.

Updated modules

mod_l10n Added timezone support.

mod_development Added dispatch debugging and explanation. Added checkbox to disable the api-service `/api/development/recompile`

mod_mqtt The scomp-live custom tag was added to *mod_mqtt* (page 301), allowing you to live update a part of the page on pubsub events. Documentation for *mod_mqtt* (page 301) has been improved and a debugging option has been added so you can see which topics are published and subscribed to.

mod_base The Zotonic logo is now included in the distribution as a Webfont, resulting in crisp logo's in the admin and on the default Zotonic status website.

New an updated filters

date An optional second argument for the timezone has been added.

date_range An optional third argument for the timezone has been added.

truncate An optional second argument is added to specify the text added where the text is truncated.

truncate_html Truncates a HTML text to a specific length, ensures that all open tags are properly closed.

New transport mechanism

The client-server communication is now based on UBF-encoded messages. It has become easier to send messages to specific pages (browser tabs) and to do bi-directional communication in general. See *Transport* (page 49) for all information on this topic.

Misc

User-defined Erlang dependencies It is now possible to add extra rebar `deps` to Zotonic, by adding them to the `zotonic.config` file.

Version-locking of dependent Erlang applications Zotonic now uses the Rebar `lock-deps` plugin to keep all included dependencies at the versions that they were at when Zotonic was released. This improves the longterm stability of the Zotonic release.

Rememberme cookie changes The *rememberme* cookie (used for automatic logon) is now based on a token instead of the user id. The token is reset if the user's password is changed. Cookies set using the previous scheme are invalidated.

Reuqest context notification Added the notification `request_context`. This is a foldl with the *Context* and is called after the request's query arguments are parsed using `z_context:ensure_qs/1`. It can be used to perform transformations or actions based on the query arguments.

Contributors

The following people were involved in this release:

Alberto López, Arjan Scherpenisse, Arthur Clemens, David de Boer, Jeff Bell, jult, Maas-Maarten Zeeman, Marc Worrell, Mawuli Adzaku and Stephan Herzog.

Release 0.11.1

Released on 2014-10-20 22:47 by arjan.

Arjan Scherpenisse (7):

- core: Disable sendfile support by default and make a note of this
- core: Fix crashing make on initial build
- core: When using locked deps, still add the deps from zotonic.config
- doc: Fix preinstall notes about buggy erlang versions
- doc: Fix sidebar
- doc: fix sidebar link
- scripts: Add "-v" argument to zotonic command to print the version

Arthur Clemens (1):

- doc: update references to priv/config

Maas-Maarten Zeeman (8):

- build: Added delete-deps
- build: Fix make clean
- build: Update mochiweb and webzmachine in .lock file
- core: Fix z_sites_dispatcher so it accepts empty paths. Fixes #842
- core: Fixes IE problems in zotonic js
- core: IE8 fixes for ubf js
- core: Prune context and spawn notifier process only when needed
- core: Remove error and close handlers before ws restarts.

Marc Worrell (31):

- admin: force to select category when adding new content.
- base: refactor the moreresults action.
- core: For websocket, keep reqdata information for 'is_ssl' checks.
- core: add acl mime check to m_media:replace/3.
- core: add lager warnings when modules are stopped.
- core: add m.modules.active.mod_foobar to test if a modules is active. Remove checks with the code server if a module is running.
- core: add mime type exception for WMA files, they were recognized as video files.
- core: add sanitization of text/html-video-embed. Move #context handling out of z_html to z_sanitize.
- core: do not delete/insert edges when changing the order via mod_menu
- core: fix concatenating certain combined file streams.
- core: lager info with modules to be started.
- core: m_rsc:p_no_acl/3 request for a non-existing resource should return 'undefined', just like m_rsc:p/3.
- core: module start/stop progress messages are now debug level.
- core: remove debug statement from m_media
- core: remove nested transaction from the z_edge_log_server check.
- erlydtl: allow lookup of var.key for a list [{<<key>>, ...}]

- filestore: use filezcache:locate_monitor/1 to let the filezcache track z_file_entry processes.
- m.modules: replace usage of m.modules.info. with m.modules.active.
- m_identity: don't crash if #identity_password_match{} doesn't match any observers.
- menu: edge menu sorter has a problem with sorting nested collections. Disable sorting for now.
- mod_admin: if date is not editable, display it as text.
- mod_admin: more generic css for .category spans.
- mod_admin: move validation inside the category_select block
- mod_admin: when replacing a media item, show the oembed/video-embed panels for embedded content.
- mod_admin_identity: prevent Safari autofilling username/passwords in new user-account forms. Fixes #811
- mod_menu/mod_admin_frontend: enable editing of collections as side-menu.
- mod_menu: fix for passing the item_template option to the server when adding items.
- mod_survey: evaluate empty jump conditions to 'true'
- mod_tkvstore: don't start as named module.
- smtp: Initialize e-mail server settings on startup. This is needed now that disc_copies are used.
- translation: added some missing nl translations.

Release 0.12.0

Welcome Zotonic 0.12.0, released on October 8, 2014.

This release has been released on the same day as *Release 0.11.0* (page 118). The main difference between these 2 releases is that in 0.12, Bootstrap has been upgraded on all templates. 0.11 was released as a feature release with all major improvements of the last 6 months excluding Bootstrap 3 support.

Bootstrap CSS version 3

Zotonic has been switched over to the latest version of the Bootstrap Framework, version 3.2.0. When you are using Zotonic's `mod_bootstrap` or when you have customized the admin templates, you will need to update your templates.

A full migration guide to upgrading from Bootstrap 2.x is here: http://getbootstrap.com/migration/, a tool to help you convert your Zotonic templates is located here: https://github.com/arjan/bootstrap3-upgrader.

Release 0.12.1

Released on 2014-10-20 17:01 by arjan.

Arjan Scherpenisse (6):

- core: Disable sendfile support by default and make a note of this
- core: Fix crashing make on initial build
- core: Use rebar locked on 0.12 as well
- core: When using locked deps, still add the deps from zotonic.config
- doc: Fix preinstall notes about buggy erlang versions
- scripts: Add "-v" argument to zotonic command to print the version

Arthur Clemens (1):

- doc: update references to priv/config

Maas-Maarten Zeeman (8):

- build: Added delete-deps
- build: Fix make clean command
- core: Fix z_sites_dispatcher so it accepts empty paths. Fixes #842
- core: Fixes IE problems in zotonic js
- core: IE8 fixes for ubf js
- core: Prune context and spawn notifier process only when needed
- core: Remove error and close handlers before ws restarts.

Marc Worrell (31):

- admin: force to select category when adding new content.
- base: refactor the moreresults action.
- core: For websocket, keep reqdata information for 'is_ssl' checks.
- core: add acl mime check to m_media:replace/3.
- core: add lager warnings when modules are stopped.
- core: add m.modules.active.mod_foobar to test if a modules is active. Major performance increase.
- core: add mime type exception for WMA files, they were recognized as video files.
- core: add sanitization of text/html-video-embed. Move #context handling out of z_html to z_sanitize.
- core: do not delete/insert edges when changing the order via mod_menu
- core: fix concatenating certain combined file streams.
- core: lager info with modules to be started.
- core: m_rsc:p_no_acl/3 request for a non-existing resource should return 'undefined', just like m_rsc:p/3.
- core: module start/stop progress messages are now debug level.
- core: remove debug statement from m_media
- core: remove nested transaction from the z_edge_log_server check.
- erlydtl: allow lookup of var.key for a list [{<<key>>, ...}]
- filestore: use filezcache:locate_monitor/1 to let the filezcache track z_file_entry processes.
- m.modules: replace usage of m.modules.info. with m.modules.active.
- m_identity: don't crash if #identity_password_match{} doesn't match any observers.
- menu: edge menu sorter has a problem with sorting nested collections. Disable sorting for now.
- mod_admin: if date is not editable, display it as text.
- mod_admin: more generic css for .category spans.
- mod_admin: when replacing a media item, show the oembed/video-embed panels for embedded content.
- mod_admin_identity: prevent Safari autofilling username/passwords in new user-account forms. Fixes #811
- mod_menu/mod_admin_frontend: enable editing of collections as side-menu.
- mod_menu: bootstrap3 change.
- mod_menu: fix for passing the item_template option to the server when adding items.
- mod_survey: evaluate empty jump conditions to 'true'
- mod_tkvstore: don't start as named module.
- smtp: Initialize e-mail server settings on startup. This is needed now that disc_copies are used.

- translation: added some missing nl translations.

Release 0.12.2

Released on 2014-12-18 17:01 by mworrell.

Major changes are:

- Addition of new notifications for importing media.
- Authentication using mod_twitter and mod_facebook is now via a popup window
- mod_twitter can now follow keywords and multiple users
- New admin menu item **Auth -> App Keys & Authentication Services**

The following commits were done

Arjan Scherpenisse (3):

- build: force rebar to build 'setup' app
- mod_admin: Fix link to query documentation
- mod_search: Use binary parsing of stored query resource

Arthur Clemens (79):

- admin: button dropdowns for admin overview filters
- admin: fix checkbox in language dropdown
- admin: fix erroneous rule
- admin: fix insert depiction dialog
- admin: fix negative margins of editor
- admin: fix nested links in table
- admin: fix nested treelist
- admin: fix tab order of form fields
- admin: formatting
- admin: handle multiple lined tree list items
- admin: hide redundant Category text
- admin: improve display of thumbnails
- admin: improve interface in various locations
- admin: increase y position of fake form fields
- admin: make long filenames visible
- admin: make muted a bit lighter
- admin: minimal design of help buttons
- admin: more horizontal forms
- admin: more reordering of less styles
- admin: multiple css fixes forms and edit blocks
- admin: NL translation fixes
- admin: prevent nesting modal-body

- admin: refactor tabs
- admin: remove @baseFont reference
- admin: remove extraneous space
- admin: remove periods from dialog headers
- admin: remove unused file
- admin: tweak admin login in release branch
- admin: use translated string
- admin: various markup improvements
- doc: add action set_class
- mod_acl_simple_roles: add titles to modules and category names
- mod_acl_simple_roles: fix div nesting
- mod_acl_simple_roles: improve modules list
- mod_acl_simple_roles: NL typo
- mod_acl_simple_roles: remove period from header
- mod_admin: allow table parts to be not clickable
- mod_admin: also update dashboard button row
- mod_admin: block menu is right aligned
- mod_admin: bootstrap 3 uses class text-muted
- mod_admin: break apart dashboard blocks for easier overriding
- mod_admin: consistent New Page dialog header
- mod_admin: css tweaks
- mod_admin: enable “All pages” button when qcat is defined
- mod_admin: handle empty cat param value
- mod_admin: improve stacked elements within widgets
- mod_admin: layout and translation of dialog Duplicate Page
- mod_admin: make button dropdown alignment configurable
- mod_admin: make white borders in media visible
- mod_admin: more horizontal forms
- mod_admin: NL typos
- mod_admin: organize less styles in separate files
- mod_admin: preserve query string when changing category
- mod_admin: prevent disappearing of media images after undo
- mod_admin: remove period from logged in user
- mod_admin: show category and query in window title
- mod_admin: show crop center in connection overview
- mod_admin: show unpublished state in connected media
- mod_admin: smaller crop center lines
- mod_admin: support em-based buttons in button row
- mod_admin: update table row hover, make colors consistent

- mod_admin: use close button without glyphicon element
- mod_admin_config: consistent field widths
- mod_authentication: remove period from header
- mod_backup: typo
- mod_backup: use bullet list for warnings
- mod_base: document params width and addclass in js source
- mod_base: fix dialog text
- mod_base: missing button style
- mod_base: remove unused line of code
- mod_base: support dialog option backdrop “static”
- mod_base: tweak debug info
- mod_editor_tinymce: add newest version
- mod_editor_tinymce: add option to always use the newest version
- mod_editor_tinymce: if no config, use newest
- mod_editor_tinymce: limit autoresize; fix resize handle bug
- mod_oembed: activate upload button
- mod_seo: optimize descriptions
- mod_signup: remove period from link

Maas-Maarten Zeeman (10):

- Changed the websocket implementation.
- core: Added recon application
- core: Fix: return default when there is no session.
- deps: Updated mochiweb, ip-log fix for R15
- deps: upgraded sendfile
- deps: Upgraded webzmachine and mochiweb
- mod_base: More careful websocket handshake.
- mod_base: Removed commented out code.
- mod_seo: Make noindex and notrack configurable from templates
- Removed comment to fix edoc generation

Marc Worrell (47):

- Add docs
- core: add exceptions for .xls and .xlsx files to z_media_identify. Fixes #893
- core: added comment explaining expire_1 and expire_n for sessions. Issue #881
- core: allow a non-integer category id to be passed to all_flat/2
- core: allow setting any rsc property that is ‘undefined’ to ‘false’.
- core: ensure all db timestamp columns have a time zone.
- core: fix args for transport ack.
- core: fix problem where mod_signed_url could not keep the user logged on.
- core: fix problem where the custom redirects form was not saved

- core: fix specs in z_db.
- core: make session cookie name configurable (solves problems where old cookies might interfere, especially on Chrome)
- core: on context prune-for-scomp, leave an empty list for request headers. Normalize user-agent lookup.
- core: removed debug from z_pivot_rsc
- core: the {% script %} tag has now arguments.
- core: z_search: fix acl check sql query.
- Create database when starting site
- docs: adapt docs to changes in files.
- install: use openssl to generate the admin password, as tr/urandom combo hangs on OS X. Fixes #847
- Make menu_subtree compatible with names
- media/embed: fixes for twitter streaming, added notifications for importing and analyzing fetch url media-data.
- mod_admin/mod_admin_frontend: preparations to allow creation of resources via the edit page.
- mod_admin: remove api dispatch rule, also defined in mod_base.
- mod_admin: return the correct context in controller_admin_media_preview
- mod_admin_frontend: fix a problem where combining the nestedSortable.js lib with other js files will result in errornous drag behaviour
- mod_authentication: export send_reminder/2 and lookup_identities/2.
- mod_authentication: fix logon_box form input "password"
- mod_authentication: Refactor twitter/facebook logon and signup code.
- mod_base: do not redirect if redirect id is set to undefined
- mod_base: fix js error in livevalidation.
- mod_base: for catinclude, don't assign passed categories to 'id'. Only assign a resource id to id.
- mod_base: in do_popupwindow use e.preventDefault() to play nice with multiple click event listeners.
- mod_base: remove extra from phone/_navbar
- mod_email_receive: when adding recipiens, catch references to non existing rsc
- mod_facebook: add delegate for saving settings.
- mod_menu/admin_frontend: final fix for new-page topic on menu insertion.
- mod_menu: correct pubzub.publish topic.
- mod_menu: remove console.log message.
- mod_mqtt: fix js error in for loops.
- mod_mqtt: fix problem where removing one topic listener removed all listeners. Cleanup live subscriptions for removed elemnts.
- mod_oembed: don't display the media twice in the admin
- mod_oembed: remove http: protocol from embed html, this enables content to be viewable on https: pages.
- mod_search: allow dates like 'now' and '+2 weeks' in search questions.
- mod_survey: allow printable overview if user has edit rights.
- mod_translation: add more rtl languages.
- mod_twitter: fix edoc problem.

- Remove is_integer check for cc5d94
- smtp: more relaxed error handling for spamd errors.

Release 0.12.3

Released on 2014-12-22 10:30 by mworrell.

Major changes are:

- Fix of the twerl git url in rebar.config.lock
- New z_stdlib, fixing issues with url metadata extraction
- Sanitization of oembed content
- Addition of Font Awesome 4
- Extended Zotonic logo font

The following commits were done

Arthur Clemens (20):

- admin: general layout fixes
- admin: use zotonic icons
- doc: update logo and link style
- fix gitignore
- mod_admin: remove unused files
- mod_admin: update with mod_base changes
- mod_artwork: add font awesome 4
- mod_artwork: add zotonic logos
- mod_artwork: rename logo folder to “zotonic”
- mod_base: allow other libs to import z.icons.less
- mod_base: bring back (predictable) circle icons
- mod_base: extend logo font to include general icons for Zotonic modules
- mod_base: make extra tag more compatible
- mod_base: replace some icons with FA versions, add button styles
- mod_base: simplify icon creation; add extending FA icons
- mod_base: update favicon
- Remove unused files
- zotonic_status: add less files
- zotonic_status: make responsive, update appearance
- zotonic_status: notification tweaks

Marc Worrell (8):

- core: add instagram js and urls to whitelist.
- core: lock new z_stdlib library. Fix twerl git url. Fixes #895
- mod_admin: always show media content, independent if size was defined.

- mod_oembed/mod_twitter: prefer our own 'website' extraction above oembed links. Pass the tweet url in the import_resource notification
- mod_oembed: add missing fallback _oembed_embeddable.tpl
- mod_oembed: sanitize received oembed code html and texts.
- mod_survey: fix problem where displaying the results did not work due to move sanitization functions.
- mod_twitter: fix import of tweets with special-html chars, was double escaped in title.

Release 0.12.4

Released on 2015-02-20 14:19 by arjan.

Arjan Scherpenisse (13):

- core: Add 'preview_url' to rsc export API call
- core: Fix edocs build
- doc: Clarify that finished/upcoming filters don't perform sorting
- mod_acl_simple_roles: Fix ACL check when category of rsc cannot be found
- mod_development: Enable automatic recompilation on MacOS X
- mod_search: API: Add limit, offset, format arguments to search API
- mod_search: Add 'finished' search filter + documentation
- mod_search: Add {finished} search method
- mod_survey: Add 'submit' API service
- mod_twitter: Close login window when user denies login request
- mod_twitter: Set all context vars when rendering logon_done template
- mod_twitter: Show logon page in current context's language
- mod_video: Make command line to ffmpeg/ffprobe calls configurable

Arthur Clemens (44):

- mod_admin: remove confusing text
- Social login: use FB compliant "Login with..."
- admin: fix zlink from editor, remove superseded template
- admin: include font-awesome version 4
- admin: tidy up Authentication Services page
- basesite: remove old fix for logo image
- doc: Document template models
- doc: add icons documentation
- doc: document 'page_url with' syntax
- doc: document more use cases for if
- doc: formatting
- doc: remove deprecated %stream%
- doc: restructure actions
- doc: restructure filters
- doc: reword solution

- doc: show css output when extending icons
- mod_admin: cosmetic fixes logon widget
- mod_admin: fix layout person form
- mod_admin: include icons css instead of recreating with less
- mod_admin: layout tweaks
- mod_admin: mobile tweaks
- mod_admin: tidy email table
- mod_base: add non-circle icons
- mod_base: add share icon
- mod_base: add social login icons
- mod_base: clean up documentation
- mod_base: hide original x char
- mod_base: include all font source files
- mod_base: make Z icons independent of FontAwesome
- mod_base: make icon extend cleaner
- mod_base_site: add icon css
- mod_base_site: better defaults
- mod_bootstrap: version 3.3.2
- mod_development: flush when translation file changes
- mod_facebook: typo
- mod_filestore: tidy up html
- mod_filestore: use 0 if archive size is undefined
- social login: use icons and update brand colors
- translation tweaks (NL)
- validation: provide message_after param to templates

David de Boer (1):

- Connect to “postgres” when creating database

Maas-Maarten Zeeman (4):

- core: Make sure the z_file_entry fsm uses correct timeouts
- core: prevent reconnecting a ws when the page is unloading
- mod_admin_identity: Fix for adding email addresses
- mod_base: Close websocket when unloading page. Fixes #898

Marc Worrell (67):

- New z_stdlib locked.
- core: add ‘expected’ option to m_rsc:update.
- core: add compile/0 and /1 to z.erl, for compiling without flush.
- core: added e.issuu.com and static.issuu.com to the sanitizer whitelist.
- core: allow binaries for some special keys.
- core: better handling of errornous urls for the z_file/media routines.

- core: correct the language utf8 encoding for R16+
- core: correctly parse multipart/signed emails.
- core: extra utf8 sanitization of received email's subject, text, html and from.
- core: fix for importing structured blocks (like during imports)
- core: fix in m_identity where fetching email identities could loop on a check if the email property was known as an identity.
- core: fix problem in m_rsc:update where modified was not set on save.
- core: fix problem where erlydtl_runtime crashed on fetching a value from a 'time_not_exists' atom.
- core: fix stacktrace shown in transport lager messages.
- core: m_rsc:update now converts non-tuple dates and handles creator/modified on import correctly.
- core: move erlang:get_stacktrace() outside of lager calls. Otherwise a stacktrace of lager will be shown due to the parse transforms.
- core: on startup z_dropbox moves now all processing files to unhandled.
- core: refactor database creation on site init.
- core: remove unreachable code.
- core: set the edge's creator_id on insert
- core: truncate the slug at 78 characters.
- core: z_datetime:to_datetime/1 now also handles numerical timestamps.
- docs: add link to the pdf version.
- docs: added placeholders.
- docs: correct the {% call %} documentation.
- m_identity: fix spec of get_rsc_types/2
- mod_admin: add pubzub and some related javascripts. needed for live tags etc.
- mod_admin: also log stacktrace on a catch.
- mod_admin: fix a problem where quick-editing a rsc adds all enabled languages.
- mod_admin_identity: publish identity changes to the topic ~/rsc/1234/identity.
- mod_admin_identity: some extra padding for the identity verification page.
- mod_admin_identity: typo in translation.
- mod_authentication/mod_twitter/etc: changes for new font-awesome, bs3 and some small tpl fixes
- mod_authentication: add authentication via LinkedIn. Add possibility to connect/disconnect accounts with FB/LinkedIn/Twitter. Fix redirects after using an external service for authentication. List connected authentication services in the password reminder email.
- mod_authentication: add special error message if there are cookie problems and the current browser is Safari 8. Issue #902
- mod_authentication: make logon form responsive, add optional page_logon with title/body texts.
- mod_base: added filter trans_filter_filled/3 export.
- mod_base: added the filter 'trans_filter_filled'
- mod_base: check dialog height repeatingly, account for rounding errors in height calculation.
- mod_base: filter-sort of undefined is undefined.
- mod_base: handle ping/pong websocket control frames, remove name conflict with zotonic ping/pong.

- mod_import_csv/core: fixes for importing categories, new properties, corrected basename in #import_csv_definition{}
- mod_import_csv: added checks to the model creation.
- mod_import_csv: fix handling of blocks. Add support for 'blocks.name.field' keys in m_rsc:update
- mod_import_csv: fixes for file handling and medium_url imports.
- mod_import_csv: major changes to mod_import_csv.
- mod_instagram: authenticate and import tags from Instagram
- mod_instagram: fix property name in comment.
- mod_l10n: adaptations for utf8 parsing changes in R17
- mod_l10n: add utf-8 encoding hints to source file
- mod_linkedin: modify template for bootstrap3
- mod_linkedin: seems LinkedIn doesn't like URL encoded secrets?
- mod_linkedin: try to workaround a problem where LinkedIn doesn't recognize the Access Token it just handed out.
- mod_linkedin: work around for a problem with access-tokens at linkedin.
- mod_mqtt: allow topics like ['~site', 'rsc', 1234].
- mod_oembed/mod_video_embed: fix problem with access rights if new media insert was done without admin rights.
- mod_oembed: don't crash on oembed connect timeouts.
- mod_signup: show external auth services for signup using the logon methods. Also always force the presence of an username_pw identity for signed up users.
- mod_survey: fix 'stop' survey button.
- mod_twitter: Fix twitter redirect url
- rebar.config.lock: lock new z_stdlib
- rebar.config.lock: new s3filez
- rebar.config.lock: new z_stdlib
- rebar.config.lock: new z_stdlib
- rebar.config.lock: new z_stdlib
- rebar.config.lock: new z_stdlib
- skel: add mod_mqtt to the base site, as it is needed by mod_admin

(2):

- emqtt_auth_zotonic issue would cause crashed when mqtt client try to connect onto it.
- Fix the emqtt client connection issue.

Release 0.13.0

Welcome Zotonic 0.13.0, released on July 3, 2015.

Major changes are:

- New ACL module: mod_acl_user_groups (see below)
- New module: mod_content_groups
- New model: m_hierarchy, for category, content_group and user_group hierarchies

- Category renumbering is now a lot faster and more incremental
- Addition of Material Design art work
- New module: mod_email_status
- New module: mod_component
- DNSBL checks for incoming emails
- More strict sanitization of content
- Social integration with Twitter, Facebook, LinkedIn, and Instagram
- *is_dependent* Resources
- Refactored automatic compile and load of changed files

This release also incorporates all fixes in the 0.12.x branch.

A complete *git shortlog* since the 0.12.0 release is added below these release notes.

Documentation

Unfortunately we still miss some documentation for the new modules. This will be corrected as soon as possible.

New Access Control module

The new *mod_acl_user_groups* module adds rule based access control.

- All resources (pages) are assigned a content group.
- All users are member of zero or more user groups.
- Content groups are arranged in an hierarchy
- User groups are arranged in an hierarchy

Rules are defined between the user groups and the content groups. Per rule the following properties can be set:

- User group
- Content group
- Category (or *all categories*)
- Privileges: view, edit, link, delete
- Deny flag (for a negative rule)

The collection of rules has an *edit* and *publish* version. The *edit* version can be tested with a special url. If all is accepted, then the *edit* version van be published.

The *edit* version can also be exported and imported. This includes the full hierarchies of all user- and content groups.

Categories and m_hierarchy

The category hierarchy tables have been replaced by *m_hierarchy*. This model defines named hierarchies of resources (pages).

If the categories are changed then the system needs to update the *pivot_category_nr* field of all resources. With the introduction of *m_hierarchy* this renumbering is much more efficient and will only affect a minimal number of resources.

The *m_hierarchy* module is also used for the content- and user group hierarchies, as used by the new *mod_acl_user_groups* module.

Email status

The new module *mod_email_status* tracks for all outgoing email addresses:

- If emails are successfully sent
- Number of emails sent
- Number of errors
- Number of bounces
- Latest error message

With this it will be much easier to get feedback on all outgoing email.

DNSBL checks

All incoming SMTP connections are now checked against two DNS block lists:

- `zen.spamhaus.org` (http://www.spamhaus.org/zen/)
- `dnsbl.sorbs.net` (http://dnsbl.sorbs.net/general/using.shtml)

The list of DNSBL servers checked can be configured using the `smtp_dnsbl` key in the `zotonic.config` file.

mod_component

This is an experimental module. It will be the basis of a new method of loading *components* in HTML pages. Components consist of HTML, javascript, and css. These parts can be dynamically loaded. Layout can/will be done using *mithril.js*.

This module will undergo significant changes, so use at your own risk. It is included to support some parties that use this in production.

Sanitization of content

`iframe` Elements and CSS are now sanitized. The *iframes* are sanitized using a whitelist of servers. Use the `#sanitize_embed_url{}` notification to add or remove servers from the whitelist. The current whitelist can be seen at the bottom of the `src/support/z_sanitize.erl` file.

Object tags referring to services like *youtube* are replaced with *iframe* tags.

This sanitization is also done on all *embed* codes of media items.

The CSS parser is strict and only allows well formed CSS. It also strips CSS that is known to be (potentially) *dangerous*:

- loading external content
- `position: screen`
- some classes that are used internally in Zotonic
- and some more

Check `deps/z_stdlib/src/z_css.erl` for the actual sanitizer code.

Social integration

The integration with social sites is completely redone. It is now possible to login (and signup) with Facebook, Twitter, LinkedIn, and Instagram.

It is also possible to import content from Twitter and Instagram. For this it is possible to define tags and/or users to follow. All matching content is imported into special categories. An optional image or embed code is also imported and added *as part of* the imported resource (so not as a separate depiction).

is_dependent Resources

A new flag is added to all resources: *is_dependent*

If this flag is set, and a connection to the resource is deleted, then Zotonic will check if there are any other *incoming* connections to the resource. If not then the resource will automatically be deleted.

This makes it possible to have resources (images, documents) that can only exist in the context of another resource. If that referring resource is deleted then the depending resources are deleted as well.

Automatic compile and load of changed files

The core system now starts either `inotify-tools` or `fswatch`, depending on which one is available. You have to install one of these to enable auto-compile and auto-load of changed files.

These are watching for any change to files in Zotonic and the sites.

If a file is changed then Zotonic will:

- Recompile if a .erl file changed
- Regenerate css if a .sass, .less or .scss file changed
- Regenerate js if a .coffee file changed
- Reindex the module index if a lib file changed (.css, .js, etc)
- Reload translations if a .po file changed
- Reload a module if a .beam file changed
- Reload dispatch rules if a dispatch file changed

If a Zotonic module is reloaded then it will be automatically restarted it the function exports or the `mod_schema` is changed.

Commits since 0.12.0

There were 586 distinct commits (more than 600 in total) since release 0.12.0.

The committers were: Alain O'Dea, Arjan Scherpenisse, Arthur Clemens, David de Boer, Jeff Bell, Maas-Maarten Zeeman, Marc Worrell, Marco Wessel, Paul Guyot, Paul Monson, Sergei, Witeman Zheng, imagency, and .

Besides these commits many people contributed to this release.

Big thanks to all of you!

Git shortlog

For clarity, some cherry-picks and development branch merges are removed from this list.

Alain O'Dea (2):

- Fix#891

- scripts: Fix activating core / user modules

Arjan Scherpenisse (63):

- core: Move master up to 0.13-dev
- mod_base_site: Small Bootstrap 3 fixes
- core: Fix crashing make on initial build
- core: When using locked deps, still add the deps from zotonic.config
- scripts: Add "-v" argument to zotonic command to print the version
- doc: Fix preinstall notes about buggy erlang versions
- core: Disable sendfile support by default and make a note of this
- Add release notes for 0.12.1
- Add release notes for 0.11.1
- Merge pull request #856 from Yozhig/patch-1
- mod_search: Use binary parsing of stored query resource
- Merge pull request #857 from Yozhig/master
- mod_base: Add JSON source of the Zotonic logo font
- build: Add 0.11 and 0.12 release branches to travis
- build: Try to coerce Travis-CI not to run rebar get-deps by itself
- core: Prefix z_db error messages with site name
- mod_admin: Fix link to query documentation
- mod_acl_simple_roles: Fix ACL check when category of rsc cannot be found
- mod_twitter: Close login window when user denies login request
- mod_development: Enable automatic recompilation on MacOS X
- mod_twitter: Set all context vars when rendering logon_done template
- mod_twitter: Show logon page in current context's language
- mod_survey: Add 'submit' API service
- mod_video: Make command line to ffmpeg/ffprobe calls configurable
- core: Add 'preview_url' to rsc export API call
- mod_search: API: Add limit, offset, format arguments to search API
- core: use epgsql 2.0.0 instead of master
- mod_search: Add 'finished' search filter + documentation
- core: Fix edocs build
- mod_search: Add {finished} search method
- doc: Clarify that finished/upcoming filters don't perform sorting
- doc: Fix sphinx configuration
- Add release notes for 0.12.4
- mod_survey: Month names in dutch are lower case
- mod_content_groups: Put in 'structure' admin menu; fix dutch wording
- core: Remove category information from rsc export
- mod_content_groups: Put content group category in 'meta'

- mod_content_groups: Add .nl translation strings
- mod_acl_user_group: Initial version
- mod_search: Allow filtering resources on content group
- Add acl rule model
- Add ACL rule editor for module and rsc rules
- mod_survey: Add 'allow_missing' request parameter
- mod_admin: Place upload form in a block, allowing it to be easier overruled
- mod_admin: (CSS) file input buttons have varying heights on different OSs
- build: Bump Travis OTP version
- m_hierarchy: Add parents/3 function
- mod_acl_user_groups: Make ACL rules overview more usable
- Add edit basics dialog for user with tab to set user groups + options
- Let search box work on users page
- Search placeholder text
- Users overview: show all persons/institutions or only those with accounts
- Create import/export function for rules
- Fix compilation error in ACL rule export controller
- mod_development: Don't let sass create a source mapping
- mod_development: Don't let sass create a source mapping
- doc: Remove reference to Ubuntu PPA
- mod_survey: Fix results/printable page when survey rsc has page path
- mod_admin_identity: Remove big 'show all' button when searching users
- mod_admin_identity: Start with showing only users, not persons
- core: z_media_preview: Do not add white space to small images when resizing
- mod_base: Allow to set response headers from service API calls
- mod_base: Correct ReqData attribute in API controller on processing POST

Arthur Clemens (160):

- doc: update references to priv/config
- admin: fix negative margins of editor
- admin: make long filenames visible
- admin: button dropdowns for admin overview filters
- admin: remove @baseFont reference
- admin: remove unused file
- admin: fix nested links in table
- admin: prevent nesting modal-body
- admin: improve display of thumbnails
- admin: hide redundant Category text
- admin: increase y position of fake form fields
- admin: NL translation fixes

- admin: use translated string
- mod_admin: bootstrap 3 uses class text-muted
- admin: make muted a bit lighter
- mod_acl_simple_roles: improve modules list
- admin: fix tab order of form fields
- admin: improve interface in various locations
- admin: handle multiple lined tree list items
- mod_signup: remove period from link
- mod_authentication: remove period from header
- mod_admin: organize less styles in separate files
- mod_admin: block menu is right aligned
- admin: remove extraneous space
- admin: more reordering of less styles
- mod_seo: optimize descriptions
- admin: fix checkbox in language dropdown
- mod_backup: use bullet list for warnings
- mod_backup: typo
- mod_editor_tinymce: add newest version
- mod_editor_tinymce: add option to always use the newest version
- mod_editor_tinymce: limit autoresize; fix resize handle bug
- mod_admin: NL typos
- mod_base: missing button style
- mod_admin: break apart dashboard blocks for easier overriding
- mod_admin: css tweaks
- mod_admin: remove period from logged in user
- mod_editor_tinymce: if no config, use newest
- admin: multiple css fixes forms and edit blocks
- admin: multiple css fixes forms and edit blocks
- mod_admin: prevent disappearing of media images after undo
- admin: formatting
- mod_admin: use close button without glyphicon element
- mod_admin: more horizontal forms
- mod_acl_simple_roles: remove period from header
- mod_oembed: activate upload button
- mod_admin: make button dropdown alignment configurable
- mod_acl_simple_roles: add titles to modules and category names
- mod_base: fix dialog text
- admin: remove periods from dialog headers
- admin: minimal design of help buttons

- mod_acl_simple_roles: fix div nesting
- mod_admin: allow table parts to be not clickable
- mod_admin: update table row hover, make colors consistent
- admin: more horizontal forms
- mod_acl_simple_roles: NL typo
- admin: refactor tabs
- mod_base: document params width and addclass in js source
- mod_admin: preserve query string when changing category
- mod_base: tweak debug info
- mod_admin: improve stacked elements within widgets
- mod_authentication: refactor logon templates
- admin: fix insert depiction dialog
- admin: various markup improvements
- admin: fix nested treelist
- mod_admin: layout and translation of dialog Duplicate Page
- mod_admin: handle empty cat param value
- mod_admin: enable “All pages” button when qcat is defined
- mod_admin: consistent New Page dialog header
- mod_admin_config: consistent field widths
- mod_admin: also update dashboard button row
- doc: add action set_class
- mod_base: remove unused line of code
- mod_admin: show unpublished state in connected media
- mod_admin: make white borders in media visible
- mod_admin: show crop center in connection overview
- mod_admin: smaller crop center lines
- mod_base: support dialog option backdrop “static”
- mod_admin: support em-based buttons in button row
- mod_artwork: add font awesome 4
- mod_artwork: add zotonic logos
- doc: update logo and link style
- Merge pull request #896 from pmonson711/readme_fixes
- mod_artwork: rename logo folder to “zotonic”
- mod_base: extend logo font to include general icons for Zotonic modules
- admin: use zotonic icons
- mod_admin: remove unused files
- admin: general layout fixes
- zotonic_status: make responsive, update appearance
- fix gitignore

- zotonic_status: add less files
- mod_base: update favicon
- Remove unused files
- mod_base: simplify icon creation; add extending FA icons
- mod_base: replace some icons with FA versions, add button styles
- mod_admin: update with mod_base changes
- zotonic_status: notification tweaks
- mod_base: make extra tag more compatible
- mod_base: allow other libs to import z.icons.less
- mod_base: bring back (predictable) circle icons
- doc: typo
- mod_base: add non-circle icons
- mod_base: make icon extend cleaner
- mod_admin: mobile tweaks
- mod_base: hide original x char
- doc: restructure actions
- admin: include font-awesome version 4
- doc: typo
- mod_base_site: add icon css
- doc: sort lines
- doc: restructure filters
- basesite: remove old fix for logo image
- mod_filestore: use 0 if archive size is undefined
- mod_filestore: tidy up html
- Don't crash when email is undefined
- doc: formatting
- doc: Document template models
- mod_development: flush when translation file changes
- doc: remove deprecated %stream%
- mod_authentication: document template structure
- mod_authentication: remove superseded templates
- mod_authentication: better defaults for social login buttons
- mod_authentication: doc tweak
- mod_facebook: typo
- admin: tidy up Authentication Services page
- admin: fix zlink from editor, remove superseded template
- mod_admin: fix layout person form
- mod_admin: tidy email table
- Social login: use FB compliant "Login with..."

- mod_base: make Z icons independent of FontAwesome
- mod_base: include all font source files
- mod_base: add social login icons
- social login: use icons and update brand colors
- doc: add icons documentation
- mod_admin: remove confusing text
- mod_admin: layout tweaks
- translation tweaks (NL)
- mod_authentication: make login work in dialog
- mod_base: clean up documentation
- mod_admin: include icons css instead of recreating with less
- mod_authentication: shorter messages
- mod_authentication: layout tweaks
- mod_authentication: let admin logon use modal screens
- mod_authentication: default no box
- mod_base: add share icon
- mod_video_embed: add bootstrap responsive class
- mod_base_site: better defaults
- Revert "mod_video_embed: add bootstrap responsive class"
- doc: document 'page_url with' syntax
- doc: document more use cases for if
- mod_bootstrap: version 3.3.2
- doc: show css output when extending icons
- validation: provide message_after param to templates
- Authentication and signup: template refactoring
- doc: reword solution
- mod_artwork: add Material Design icons
- Fix log in verb
- mod_bootstrap: update update script
- mod_bootstrap: fix sourceMappingURL replacement
- mod_bootstrap: update to v3.3.4

David de Boer (10):

- Make menu_subtree compatible with names
- Remove is_integer check for cc5d94
- Create database when starting site
- Add docs
- Connect to "postgres" when creating database
- Fix page block links
- Fix modal signup

- mod_base: Allow to return 401 from API services
- mod_base: Decode JSON body in service calls
- Allow post-login redirect to be passed to logon modal

Jeff Bell (3):

- mod_admin: Flushed out ability to save [object, predicate] pairs when creating new rsc using the dialog_new_rsc action.
- CONTRIBUTORS: added myself to CONTRIBUTORS file
- mod_admin: action_admin_dialog_new_rsc fix

Maas-Maarten Zeeman (43):

- core: Fixes IE problems in zotonic js
- core: IE8 fixes for ubf js
- build: Added delete-deps
- core: Prune context and spawn notifier process only when needed
- core: Remove error and close handlers before ws restarts.
- build: Fix make clean
- core: Fix z_sites_dispatcher so it accepts empty paths. Fixes #842
- core: Added recon application
- mod_base: Removed commented out code.
- mod_seo: Make noindex and notrack configurable from templates
- Changed the websocket implementation.
- Removed comment to fix edoc generation
- mod_base: More careful websocket handshake.
- core: Fix: return default when there is no session.
- core: Export function to create a checksummed url for css and js lib files.
- mod_component: Added new module to make and use components on your page.
- fix for lazy loading css
- core: prevent reconnecting a ws when the page is unloading
- mod_base: Close websocket when unloading page. Fixes #898
- core: Fix a problem with mapping topic names to local names.
- Merge pull request #909 from witeman/emqtt_auth_issue
- mod_component: Fix bug in initilizing an already loaded component. Typo
- core: Make sure the z_file_entry fsm uses correct timeouts
- mod_component: delegate onunload to the component controller
- mod_component: updated mithril.js
- mod_component: Ignore load requests for components not in pending state
- z_utils: Allow trusted js values.
- mod_mqtt: Add an ack callback to subscribe calls.
- core: Fix for postback triggered by mqtt events.
- mod_admin_identity: Fix for adding email addresses

- core: Removed compile warning
- mod_component: Make it possible to use iolist init scripts
- core: Cache generated link and script tags when mod_development is disabled
- core: Do a session_context fold to get the right values in the context. Fixes language problem in websockets.
- core: Make acceptor_pool_size configurable. Fixes #923
- core: change default number of acceptors to 75.
- mod_acl_simple_roles: Fix to prevent blanking out all acl settings when role rsc is changed.
- mod_base_site: Make configuring the site's navbar logo easier
- mod_base_site: Minor base template fixes.
- filewatcher: fix, file_changed/2 moved
- mod_base_site: Fixed typo
- mod_base: Make synchronous ajax requests when the page is unloading
- mod_search: Make it possible to pass rsc_lists as parameter to hasanyobject.

Marc Worrell (280):

- smtp: Initialize e-mail server settings on startup. This is needed now that disc_copies are used.
- mod_survey: evaluate empty jump conditions to 'true'
- mod_menu: fix for passing the item_template option to the server when adding items.
- core: do not delete/insert edges when changing the order via mod_menu
- core: remove nested transaction from the z_edge_log_server check.
- mod_admin: if date is not editable, display it as text.
- mod_admin: more generic css for .category spans.
- mod_menu/mod_admin_frontend: enable editing of collections as side-menu.
- mod_menu: bootstrap3 change.
- core: add mime type exception for WMA files, they were recognized as video files.
- core: remove debug statement from m_media
- core: fix concatenating certain combined file streams.
- filestore: use filezcache:locate_monitor/1 to let the filezcache track z_file_entry processes. Fix difference in keys for uploading and * downloading. Better debug/info messages.
- mod_admin_identity: prevent Safari autofilling username/passwords in new user-account forms. Fixes #811
- translation: added some missing nl translations.
- erlydtl: allow lookup of var.key for a list [{<<key>>, ...}]
- core: m_rsc:p_no_acl/3 request for a non-existing resource should return 'undefined', just like m_rsc:p/3.
- core: For websocket, keep reqdata information for 'is_ssl' checks. When pruning for contexts, keep socket info for the is_ssl check.
- core: add sanitization of text/html-video-embed. Move #context handling out of z_html to z_sanitize.
- core: add acl mime check to m_media:replace/3.
- mod_admin: when replacing a media item, show the oembed/video-embed panels for embedded content.
- base: refactor the moreresults action.
- m_identity: don't crash if #identity_password_match{} doesn't match any observers.
- core: add lager warnings when modules are stopped.

- core: lager info with modules to be started.
- mod_tkvstore: don't start as named module.
- admin: force to select category when adding new content.
- menu: edge menu sorter has a problem with sorting nested collections. Disable sorting for now.
- core: module start/stop progress messages are now debug level.
- core: add m.modules.active.mod_foobar to test if a modules is active. Remove checks with the code server if a module is running.
- m.modules: replace usage of m.modules.info. with m.modules.active.
- core: on context prune-for-scomp, leave an empty list for request headers. Normalize user-agent lookup.
- core: fix args for transport ack.
- mod_email_receive: when adding recipiens, catch references to non existing rsc
- core: make session cookie name configurable (solves problems where old cookies might interfere, especially on Chrome)
- Merge pull request #864 from driebit/fix-menu-subtree-name
- Merge pull request #865 from driebit/remove-is-int-menu-subtree
- core: z_search: fix acl check sql query.
- mod_survey: allow printable overview if user has edit rights.
- mod_base: remove extra from phone/_navbar
- mod_admin: remove api dispatch rule, also defined in mod_base.
- mod_base: for catinclude, don't assign passed categories to 'id'. Only assign a resource id to id.
- mod_menu: remove console.log message.
- core: allow a non-integer category id to be passed to all_flat/2
- mod_admin/mod_admin_frontend: preparations to allow creation of resources via the edit page.
- core: allow setting any rsc property that is 'undefined' to 'false'.
- core: ensure all db timestamp columns have a time zone.
- mod_menu: correct pubzub.publish topic.
- mod_admin_frontend: fix a problem where combining the nestedSortable.js lib with other js files will result in errornous drag behaviour
- mod_menu/admin_frontend: final fix for new-page topic on menu insertion.
- core: removed debug from z_pivot_rsc
- core: fix problem where the custom redirects form was not saved
- core: fix problem where mod_signed_url could not keep the user logged on.
- mod_mqtt: fix problem where removing one topic listener removed all listeners. Cleanup live subscriptions for removed elemnts.
- core: added comment explaining expire_1 and expire_n for sessions. Issue #881
- Merge pull request #880 from witeman/mod_authentication_logon_display_bugfix
- smtp: more relaxed error handling for spamd errors.
- install: use openssl to generate the admin password, as tr/urandom combo hangs on OS X. Fixes #847
- mod_base: do not redirect if redirect id is set to undefined
- mod_base: in do_popupwindow use e.preventDefault() to play nice with multiple click event listeners.

- mod_mqtt: fix js error in for loops.
- core: the {% script %} tag has now arguments.
- mod_authentication: Refactor twitter/facebook logon and signup code.
- mod_facebook: add delegate for saving settings.
- mod_base: fix js error in livevalidation.
- mod_authentication: export send_reminder/2 and lookup_identities/2.
- Merge pull request #872 from driebit/auto-create-db
- docs: adapt docs to changes in files.
- core: fix specs in z_db.
- mod_oembed: remove http: protocol from embed html, this enables content to be viewable on https: pages.
- core: add exceptions for .xls and .xlsx files to z_media_identify. Fixes #893
- media/embed: fixes for twitter streaming, added notifications for importing and analyzing fetch url media-data.
- mod_search: allow dates like 'now' and '+2 weeks' in search questions.
- mod_admin: return the correct context in controller_admin_media_preview
- mod_translation: add more rtl languages.
- mod_twitter: fix edoc problem.
- doc: add mod_component.
- mod_oembed: don't display the media twice in the admin
- docs: added 0.12.2 release notes
- mod_oembed: add missing fallback _oembed_embeddable.tpl
- mod_oembed/mod_twitter: prefer our own 'website' extraction above oembed links. Pass the tweet url in the import_resource notification
- mod_twitter: fix import of tweets with special-html chars, was double escaped in title.
- mod_admin: always show media content, independent if size was defined.
- mod_oembed: sanitize received oembed code html and texts.
- core: add instagram js and urls to whitelist.
- mod_survey: fix problem where displaying the results did not work due to move sanitization functions.
- core: lock new z_stdlib library. Fix twerl git url. Fixes #895
- docs: fix length of header-underline
- docs: add 0.12.3 release notes
- Merge pull request #897 from driebit/connect-postgres-db
- core: refactor database creation on site init.
- mod_authentication: add authentication via LinkedIn. Add possibility to connect/disconnect accounts with FB/LinkedIn/Twitter. Fix * redirects after using an external service for authentication. List connected authentication services in the password reminder email.
- mod_linkedin: modify template for bootstrap3
- m_identity: fix spec of get_rsc_types/2
- mod_admin_identity: some extra padding for the identity verification page.
- mod_authentication: add optional page_logon for logon-title and an alert box on the logon page.

- mod_authentication: add special error message if there are cookie problems and the current browser is Safari 8. Issue #902
- mod_signup: show external auth services for signup using the logon methods. Also always force the presence of an username_pw identity for * signed up users.
- mod_linkedin: seems LinkedIn doesn't like URL encoded secrets?
- mod_admin: also log stacktrace on a catch.
- mod_oembed/mod_video_embed: fix problem with access rights if new media insert was done without admin rights.
- core: set the edge's creator_id on insert
- mod_survey: fix 'stop' survey button.
- core: fix stacktrace shown in transport lager messages.
- core: move erlang:get_stacktrace() outside of lager calls. Otherwise a stacktrace of lager will be shown due to the parse transforms.
- mod_twitter: Fix twitter redirect url
- mod_admin: add pubzub and some related javascripts. needed for live tags etc.
- mod_authentication/mod_twitter/etc: changes for new font-awesome, bs3 and some small tpl fixes
- mod_oembed: don't crash on oembed connect timeouts.
- mod_instagram: authenticate and import tags from Instagram
- core: fix problem where erlydtl_runtime crashed on fetching a value from a 'time_not_exists' atom.
- core: truncate the slug at 78 characters.
- core: fix in m_identity where fetching email identities could loop on a check if the email property was known as an identity.
- mod_base: handle ping/pong websocket control frames, remove name conflict with zotonic ping/pong.
- mod_linkedin: try to workaround a problem where LinkedIn doesn't recognize the Access Token it just handed out.
- mod_linkedin: work around for a problem with access-tokens at linkedin.
- core: allow binaries for some special keys.
- mod_import_csv: major changes to mod_import_csv.
- mod_instagram: fix property name in comment.
- mod_import_csv: added checks to the model creation.
- mod_base: check dialog height repeatingly, account for rounding errors in height calculation.
- core: add 'expected' option to m_rsc:update.
- mod_l10n: add utf-8 encoding hints to source file
- mod_l10n: adaptations for utf8 parsing changes in R17
- core: fix for importing structured blocks (like during imports)
- core: on startup z_dropbox moves now all processing files to unhandled.
- core: z_datetime:to_datetime/1 now also handles numerical timestamps.
- core: m_rsc:update now converts non-tuple dates and handles creator/modified on import correctly.
- mod_import_csv: fixes for file handling and medium_url imports.
- core: fix problem in m_rsc:update where modified was not set on save.
- mod_base: added the filter 'trans_filter_filled'

- core: remove unreachable code.
- docs: added placeholders.
- mod_import_csv: fix handling of blocks. Add support for 'blocks.name.field' keys in m_rsc:update
- core: add compile/0 and /1 to z.erl, for compiling without flush.
- docs: added xelatex target to generate PDF.
- mod_mqtt: allow topics like ['~site', 'rsc', 1234].
- mod_admin_identity: typo in translation.
- mod_admin_identity: publish identity changes to the topic ~/rsc/1234/identity.
- core: added e.issuu.com and static.issuu.com to the sanitizer whitelist.
- mod_import_csv/core: fixes for importing categories, new properties, corrected basename in #import_csv_definition{}
- doc: cleanup of pdf version.
- docs: add link to the pdf version.
- docs: better link text to the pdf version.
- docs: move the cookbooks to their own top level chapter.
- docs: correct the {% call %} documentation.
- skel: add mod_mqtt to the base site, as it is needed by mod_admin
- core: correct the language utf8 encoding for R16+
- mod_base: added filter trans_filter_filled/3 export.
- mod_admin: fix a problem where quick-editing a rsc adds all enabled languages.
- mod_base: filter-sort of undefined is undefined.
- core: correctly parse multipart/signed emails.
- core: better handling of errornous urls for the z_file/media routines.
- core: extra utf8 sanitization of received email's subject, text, html and from.
- mod_content_groups: new module to categorize content into groups for the access control.
- m_hierarchy: merge m_menu_hierarchy and m_category into m_hierarchy. m_category now interfaces to m_hierarchy.
- Merge pull request #922 from imagency/master
- Merge pull request #921 from CyBeRoni/configure-dbcreation
- core: added support for 'is_dependent' flag.
- core: set longer timeout for renumber_pivot_task query.
- core: check if rsc existed before adding a rsc_gone entry.
- mod_admin_identity: correct the button class on the identity verification page.
- mod_mailinglist: more efficient query for polling scheduled mailings.
- mod_mqtt: allow 'live' wiring postbacks to mqtt topics, depending on the presence of an element-id.
- mod_email_status: new module to track email recipient status. Changes to the email bounce, sent and failure notifications.
- docs: fix length of header underline.
- docs: add mod_email_status placeholders.
- mod_email_status: fix problem with re-installing the model.

- mod_email_status: handle non-binary error status values.
- mow_acl_user_groups: first working version. To do: docs and test interface.
- mod_menu: breack cyclic dependency [mod_acl_user_groups,mod_authentication,mod_admin,mod_menu,mod_content_grou
- Merge pull request #933 from driebit/fix-page-block-links
- Merge pull request #936 from CyBeRoni/override-pidfile
- mod_base: remove 'render-update' observer, leave template rendering to modules for more control
- mod_l10n: add some extra country-name to country-code mappings.
- mod_l10n: fix iso lookup of Serbia and Montenegro
- core: add log message when adding rsc.is_dependent and rsc.content_group_id.
- core: add is_meta/2 function to test if a category is a meta category, needed for determining the default content group in m_rsc:get/2
- mod_content_groups: use the m_category:is_meta/2 to determine the default content group.
- mod_content_groups: convert tabs to spaces.
- mod_acl_user_groups: fix search restriction if user can't see any content groups.
- core: fix progressbar for form posts with body.
- mod_content_groups: fix adding content_group_id for #rsc_get{} on an non-existing resource.
- mod_base: added the action update_iframe
- mod_email_status: add status dialog and simple templates to see the status of an email address.
- mod_base: missing part of the update_iframe addition
- mod_base: cross-platform fix for update_iframe
- mod_survey: add is_autostart option. Fix some minor template errors.
- core: use 'optional' includes for blocks.
- mod_survey: use the 'optional' include for the templates.
- core: normalize the name of the z_update_iframe JS function with its Erlang equivalent.
- mod_import_csv: Force convert to utf8 of the top row. Add test for ';' as a column separator.
- core: fix 'next' page number for queries returning a #search_result{}.
- mod_admin: use search query 'admin_overview_query' for the overview page if it is defined.
- mod_admin: add admin_content_query, also displayed in the content menu.
- mod_admin: enable 'all pages' button if a qquery is defined.
- core: ensure that the compiled template's name equals the one found via an (optional) catinclude. Issue #938
- mod_content_groups: 17.x compatibility for string with unicode character (breaks 15.x)
- mod_survey: fix crash on prep_answer if question name is not unique
- core: fix sql query in category delete.
- mod_acl_user_groups/mod_content_groups: ensure the hierarchy if the content/user groups are changed.
- mod_admin: in tree-list, show the category in gray.
- core: escape filenames using a single quot. Fixes #924
- core: shortcut for lib file lookup without filters (don't check the file store)
- core: add mod_mqtt to the default installed modules (as it is a dependency of mod_admin).
- core: fix m_category/2 lookup. Thanks to Alvaro Pagliari.

- Merge pull request #946 from AlainODea/BUGFIX_issue891
- core: fix problem where a gmail autoreply was classified as a bounce. Fix tracking of some email errors. Started collecting email test data.
- mod_email_status: change info message about previous problems.
- mod_admin: pass all strings from the new-rsc dialog form. Fixes #948
- mod_acl_user_groups: when changing category/content-group, show the meta category if the id is a meta or the current user is 1 (the admin)
- Merge branch 'master' into acl-content-groups
- zotonic_status: use default 'post' method for the logon form to prevent showing the password if there is a js error.
- mod_linkedin: adaptations for LinkedIn API changes.
- mod_linkedin: fix for picture-url.
- mod_search: fix a problem where a 'hassubject=[...]' query term was incorrectly parsed. Fixes #950
- mod_survey: change the thank you text, remove the 'Be sure to come back for other surveys' text.
- mod_search: add cat_exact query argument.
- mod_base: fix html_escape function in zotonic-1.0.js
- mod_admin_identity: on the users page, only show those with an username_pw entry. Issue #747
- mod_admin_identity: show 'email status' buttons if mod_email_status is enabled.
- mod_search: add 2nd ordering on -publication_start to featured search.
- mod_search: fix search_query for 'hasobject=123'. Fixes #953
- core: do not create atoms from rsc names in catinclude.
- core: add tick_10m and tick_6h notifications
- core: fix a problem where ImageMagick identified PDF files as PBM.
- core: cleanup location_lat/lng update/pivot code.
- mod_base: in scomp_lazy, ensure the 'visible' argument to all 'moreresults' actions
- Merge pull request #955 from pguyot/patch-2
- Merge pull request #956 from pguyot/patch-3
- mod_admin: pass all strings from the new-rsc dialog form. Fixes #948
- Merge pull request #958 from driebit/fix-signup-delegate
- core: adding some test data for smtp tests.
- Merge pull request #960 from trigeek38/fix-dialog-new-rsc
- Merge pull request #961 from trigeek38/fix-my-fix
- mod_admin_identity: fix user query.
- mod_admin: make action admin_dialog_new_rsc more robust against illegal objects arg
- mod_admin_identity: small changes/fixes to the users query.
- mod_acl_user_groups: add 'block' option to access control rules. Update the view via a 'live' subscribe.
- Start removing R15 support.
- core: move more code into the m_hierarchy update to prevent race conditions.
- core: in m_hierarchy, lock the rows of a hierarchy when updating.
- core: trace module (re-)loads and restart modules if any exported functions are changed. Issue #964

- core: bit more silent start/stop of modules.
- core: add realtime block list checks for incoming email. Needs updated z_stdlib.
- mod_development: move fswatch/inotify to the core.
- core: add cli commands 'zotonic startsitelstopsitelrestartsite'. Fixes #964
- core: fix delete of timer in fswatch.
- core: log warnings if trying to insert duplicate rsc name/uri/path
- Fixes for language handling. Allow undefined pref_language for user. Filter on valid language code on pref_language upate. Show select list * with all valid languages in /admin/translation.
- core: add m_rsc 'is_linkable' lookup, also in z_acl and m_acl
- docs: add placeholders for ACL documentation.
- mod_search: allow search terms in texts without '=value' arg, map to '=true'. Fixes #970
- [docs] proposal for site-fsm
- [core] add possibility to fetch sub-trees from a hierarchy. Example: m.hierarchy.content_group[id].tree1
- [docs] A websocket connection is opened by the browser, accepted by the server.
- mod_menu: fix issue that on insertion of a new item it is added to all sub-menus. Fixes #971
- mod_search: add query term 'hasanyobject', to search using an 'or' on outgoing edges. Fixes #968
- mod_acl_user_groups: new import/export version of the acl rules. Include the content groups, user groups and hierarchies in the export.
- mod_acl_user_groups: move ensure_name to the core m_rsc module.
- mod_menu: in the menu-editor, add open/close buttons for submenus. This makes editing big lists easier.
- docs: fix documentation of smtp server settings. Also fix z_config settings.
- core: name correction, the bounce server is a complete smtp receive server.
- docs: clarification of the bounce addresses.
- core: allow pivot task to return {delay, DateTime}.
- core: fix a problem where m_rsc:ensure_name/2 could insert duplicate names. Fixes #974
- core: add reserved usernames to prevent users signing up as these. Fixes #929
- core: if some reserved username was taken, allow to update the password. Issue #929
- core: allow diff between Date and DateTime.
- core: add 'flickrit.com' to the iframe whitelist.
- core: copy tmpfiles if they are mailed, add periodic cleanup of the tmp folder. Fixes #972
- core: fix a problem where a fulltext search with an empty category list didn't return any results. Fixes #976
- core: fix mqtt login, don't start a session_page if there is no session or reqdata. Fixes #973
- mod_admin: fix problem where the lazy-loader for moreresulsts kept loading till no results where left.
- Set version number to 0.13.0
- Lock deps
- docs: 0.13.0 release notes and some extra (minimal) documentation.
- docs: add tentatve 0.13.0 release date
- core: determine mime type of attachments if it was not given.
- core: use 'rebar' to compile zotonic from the command line. Move .yrl output to default location for rebar. > Use skip_deps for 'make compile-zotonic'. Add 'make refresh-deps' and 'make list-deps'. Fixes #978

Marco Wessel (4):

- Allow configuration of db creation and installation
- Add commented dbinstall/dbcreate options + explanation
- Add warning message that db won't be created or installed
- Allow to override pidfile location with env var

Paul Guyot (3):

- Fix typo in _block_view_survey_thurstone.tpl
- Handle &# entities when splitting markers
- mod_survey: Fix multi & single HTML structure for thurstone checkboxes

Paul Monson (1):

- Update README link

Sergei (2):

- fix dependencies build order
- force rebar to build 'setup' app

Witeman Zheng (1):

- mod_authentication: fix logon_box form input "password"

imagency (12):

- Tinymce support for Russian

(2):

- emqtt_auth_zotonic issue would cause crashed when mqtt client try to connect onto it.
- Fix the emqtt client connection issue.

Release 0.13.1

Welcome Zotonic 0.13.1, released on July 29, 2015.

This is a maintenance release of Zotonic 0.13

A complete *git shortlog* since the 0.13.0 release is added below these release notes.

Commits since 0.13.0

There were 16 commits since release 0.13.0.

The committers were: Arthur Clemens, David de Boer, Fred Pook, and Marc Worrell.

Big thanks to the committers and all other people contributing to Zotonic!

Git shortlog

Arthur Clemens (4):

- admin_development: Add hostname to dropdown to indicate this does not need to be entered
- mod_base: Use dialog width param, fix vertical centering
- Remove debugging
- doc: Document action dialog parameters

David de Boer (4):

- Make module and site includes path-independent
- Support custom redirect page after social media logon
- Make post-logon actions dynamic
- Fix opening admin dialog by removing call to dialogReposition()

Fred Pook (2):

- Added dutch translations
- Added knightlab audio plugin to the whitelist

Marc Worrell (6):

- doc: fix sidebar versions
- Merge pull request #980 from driebit/compile-include-paths
- docs: fix title attr of docs sidebar link.
- mod_search: catch errors when evaluating stored search questions. Issue #988
- mod_base: filter to_integer returns now 'undefined' for invalid inputs.
- mod_oauth: new model functions to directly add a application (with generated tokens)

Release 0.13.2

Welcome Zotonic 0.13.2, released on August 11, 2015.

This is a maintenance release of Zotonic 0.13

The main change is that this version compiles correctly on Erlang 18.

Commits since 0.13.1

There were 15 commits since release 0.13.1.

Big thanks to all people contributing to Zotonic!

Git shortlog

Marc Worrell (15):

- docs: fix ref in 0.13.1 release notes.
- mod_oauth: fix user_id initialization on direct app creation
- mod_oauth: use the abs_url of the request path for checking the signature. This fixes an issue where OAuth access to https sites is denied.
- mod_content_groups: also add the content group as related data to the pivot.
- mod_base/mod_admin: add optional redirect to the set website, using 'is_website_redirect'. Allow overruling the controller's 'is_canonical' flag with 'is_page_path_multiple'. Add documentation for rsc properties 'is_dependent', 'content_group_id', 'is_page_path_multiple' and 'is_webite_redirect'.
- New z_stdlib
- core: allow max 5 smtp connections per relay, set max parallel smtp senders to 100. Fixes #993
- mod_survey: open survey results from mod_admin_frontend in new window.
- mod_base: hide any files and directories whose name starts with a '.' Fixes #991

- core: fix compilation on Erlang 18.0. Fixes #979
- core: add erlang 18 as travis target.
- Locked new deps.
- Preliminary 0.13.2 release and release notes.
- New z_stdlib
- mod_content_groups: on pivot add the complete path for the content-group as related ids.

Release 0.13.3

Welcome Zotonic 0.13.3, released on September 9, 2015.

This is a maintenance release of Zotonic 0.13

Main changes are:

- refinements to the admin interface by Arthur Clemens
- included Bootstrap version has been upgraded to version 3.3.5
- fixes for some issues around the newly introduced content groups
- CORS support for the API (Thanks to @ghosthamlet!)
- fixes some issues with the websocket connection
- fixes an issue where the wrong language could be selected

Commits since 0.13.2

There were 103 commits since release 0.13.2.

Big thanks to all people contributing to Zotonic!

Git shortlog

Arjan Scherpenisse (1):

- mod_base: Support CORS API requests

Arthur Clemens (76):

- mod_admin: (small screen) prevent scrolling of body when menu open
- mod_admin: fix lost tree-list styling
- mod_admin: style tweaks
- mod_admin: position help buttons in widget headers
- mod_base: remove redundant div container in modal body
- Remove inadvertently created files
- mod_admin: create user dropdown in main nav
- mod_admin: improve sorting of page overview
- mod_admin: optional category-specific column
- mod_admin: update admin bootstrap
- mod_admin: prevent executing content_group query
- doc: Add dev note on translations; add batch update command

- mod_translation: Localize messages
- mod_base: make pager numbering more logical
- mod_base: update translation files
- mod_bootstrap: update to 3.3.5
- mod_base: move pager css overrides to mod_admin
- mod_admin: reorganize edit page header
- mod_admin: Make topbar elements fit on a smaller screen when country and account buttons are visible
- docs: release notes in reverse chronological order
- mod_editor_tinymce: keep text inside moved blocks
- mod_editor_tinymce: add translation files
- mod_editor_tinymce: Add latest TinyMCE 4.2.4, and updated codemirror plugin
- mod_admin: reorganize page edit header: keep image reference after saving
- mod_admin: optional category-specific column
- mod_admin: Fix unlinking media on Enter
- docs: Mobile friendly layout
- docs: Mobile friendly layout: fix search box layout
- mod_admin: revert "also check in .css files resulting from .less compilation"
- File watcher: read optional config file to run lessc with params and compile just the main less file
- mod_admin: reorganize page edit header: hide "by" if modified/creator does not exist
- mod_acl_user_groups: minor frontend tweaks
- mod_base: minor icon font tweaks
- mod_base: minor icon font tweaks
- Admin: straighten admin-header blocks
- Admin: add breadcrumbs to hierarchical pages
- File watcher: read optional config file to run lessc with params and compile just the main less file
- mod_translation: tidy up language settings dialog
- mod_admin: give logo a width to prevent jumping navbar
- docs: Add note about /etc/hosts on first site
- Fallback language
- Revert partial css files
- mod_admin: fix position of drag icon in connection item
- mod_admin: Update Dutch translations
- mod_menu: A couple of Dutch translations
- mod_menu: Improve text (and remove html markup)
- mod_menu: Fix fuzzy Dutch translations
- mod_admin_predicate: Fix string escaping in translation text
- mod_admin: Fix extraneous spaces; fix fuzzy Dutch
- mod_l10n: typo
- mod_admin: tweak layout top header

- mod_admin: tweak widget colors on edit page
- mod_admin: tweak colors dashboard
- mod_admin: remove period from “No items.”
- Fix translations by unmatched spaces in text
- mod_admin: Edit category dialog: add button to pages
- mod_admin: improve layoud edit form in dialog
- mod_admin: Fix markup in translation text
- mod_admin: tweak colors dashboard: couple of bug fixes
- mod_admin: add test query button
- mod_admin: tweak dashboard and dialog navs
- mod_translation: Give feedback when no languages can be copoed
- mod_admin: update translations
- mod_admin: update translations with msguniq
- docs: update admin layout cookbook
- mod_admin: couple of translation fixes
- mod_admin_identity: update translation files; add NL translations
- mod_admin: add button container to widget header if it is not present in the template
- Small fixes
- docs: update cookbook for dynamic select boxes
- mod_admin: hide minimize button in dialog
- mod_admin: tweak menu item style
- mod_admin: make connections list reusable
- mod_admin: make connections list reusable: fix button label, shorten add link and update translations
- mod_admin: fix dialog header translation
- docs: update info about translations

Maas-Maarten Zeeman (7):

- mod_base: Only use the websocket when it is in OPEN state.
- mod_backup: Catch errors when restoring a resource. Better user feedback in case of problems.
- authentication: Set user preferences like language and tz during a logon.
- core: Show an error when there is a problem rendering error.tpl
- core: Set reqdata in context when rendering error.tpl. Fixes #1013
- core: Make sure the error template is rendered with the expected variables and a good context.
- core: Removed error logging code on ws error handler.

Marc Worrell (19):

- mod_acl_user_groups: show default user group if user is not member of any group.
- mod_acl_user_groups: when displaying user’s groups, check for displayed user being an user.
- mod_admin: in dialog_edit_basics, only show the ‘full edit’ button if the user can access either mod_admin or mod_admin_frontend
- core: refactored pivot queue polling, now will poll faster if anything was found in the queue.

- core: force reload of client if page_id in z_msg is undefined.
- core: fix a problem where a multipart post form could loop if the end-boundary was missing
- mod_mqtt: fix race condition in re-subscribe of mqtt listeners after a module restart.
- mod_admin: also check in .css files resulting from .less compilation.
- mod_base: fix controller_api context handling in to_json.
- core: allow '1' and 'true' as timezones (both map to UTC)
- mod_admin: also check in .css files resulting from .less compilation.
- mod_base: fix a problem where a pager on page with 'id' and was not using the page_path of the resource. Fixes #1005
- core: set default timeout of sql queries higher (from 5sec to 30sec)
- mod_base: fix a problem with checking if the current path is the canonical path. Issue #1010
- mod_admin: add generated .css files for systems without less installed.
- core: change logging notifications from {log, ...} to #zlog{}. Issue #992
- core: fix typo in os:timestamp() call.
- Merge branch 'release-0.13.x' of github.com:zotonic/zotonic into release-0.13.x
- core: add 'www.google.com/maps/' to the sanitizer white list.

Release 0.13.4

Welcome Zotonic 0.13.4, released on September 16, 2015.

This is a maintenance release of Zotonic 0.13

Main changes are:

- fixed an issue where mod_survey answers could be shown without escaping
- fixes for admin css and template issues
- corrected handling of unsafe (escaped) characters in email addresses
- easier access rule definitions by special handling of 'meta' category
- fixed an issue where mod_video would crash if there was an empty ffmpeg configuration
- fixed an issue where the last row of an imported CSV file could be reversed

Commits since 0.13.3

There were 34 commits since release 0.13.3.

Big thanks to all people contributing to Zotonic!

Git shortlog

Arthur Clemens (17):

- Pass property is_dependent in datamodel; show status in edit page even if protected
- Apply tab style to all tabs within widgets
- Typo
- mod_admin: use relative date in event info

- docs: exemplify unique ids
- mod_admin: allow other modules to get result from connect dialog by passing option 'delegate'
- mod_admin: give first input field focus when switching tab
- mod_admin: improve layout of navigation bars
- mod_admin: improve layout of navigation bars: fix toolbar padding
- mod_admin: improve layout of navigation bars: improve some colors
- docs: add example of a postback action called from javascript
- mod_admin: add styling for justified tabs
- doc tweaks
- mod_admin: override style for code
- Merge remote-tracking branch 'origin/release-0.13.x' into release-0.13.x
- Revert "Merge remote-tracking branch 'origin/release-0.13.x' into release-0.13.x"
- mod_admin: override style for code: fix top bar

Marc Worrell (16):

- Merge pull request #1022 from millenaarg/release-0.13.x
- mod_import_csv: fix an issue where the last row is reversed iff there is no empty line at the end of the file.
- mod_backup: add option to make a database-only backup. Also make the backup list a live include.
- mod_backup: fix build by removing '&' from @doc
- core: fix a problem where email addresses couldn't have a ' (single quot) in them. Fixes #1023
- mod_video: use the default ffmpeg command if the configured ffmpeg command is empty.
- mod_video: use the default ffprobe and ffmpeg_preview command if the configured command is empty.
- New dependencies
- mod_acl_user_groups: ACL rules only apply to category 'meta' if either 'meta' or the system content group is mentioned in the ACL rule. Fixes #1030
- mod_l10n: added Kosovo (code xk) to the country list.
- mod_acl_user_groups: add 'view-only' option. Fixes #1034
- mod_acl_user_groups: show error if access to the ACL rules view is denied.
- mod_acl_user_groups: remove 'not' that was left in for checking.
- core: notify #hierarchy_updated observers which ids have added and/or removed.
- mod_menu: show the category in menu-list items muted.
- mod_survey/mod_base: url escape google chart arguments; escape survey label names.

millenaarg (1):

- Added reference to scomps-script

Release 0.13.5

Welcome Zotonic 0.13.5, released on October 27, 2015.

This is a maintenance release of Zotonic 0.13

Main changes are:

- Updates to the bundled tinymce and code view plugins

- Fulltext search fixes, search terms could be stemmed multiple times
- Many fixes to the admin css and html
- Fixes an ACL user groups problem where the sudo permissions didn't reflect the admin
- Fixes a problem with the template compiler where a template could be compiled multiple times
- Translation option to force the default language of a site
- Fixes for the comet and postback controllers where push data was not sent with postback requests
- New admin filter 'temporary_rsc'
- Fix for a problem in mod_video where a video could be rendered with a color space that is incompatible with QuickTime

Commits since 0.13.4

There were 101 commits since release 0.13.4.

Big thanks to all people contributing to Zotonic!

Git shortlog

Alex Popov (1):

- Fixes zotonic/zotonic#1027 issue with zotonic-module script

Arjan Scherpenisse (1):

- mod_admin: Fix media upload when no content groups configured

Arthur Clemens (41):

- mod_editor_tinymce: load the correct lib files
- mod_admin: improve reusability of connections interface
- mod_admin: fix background color of top menu on small screen
- Make fswatch work with spaces in paths
- mod_admin: prevent disconnecting when pressing enter
- mod_admin: optimize display of connection list
- doc: doc tweak
- mod_admin: tweaks
- mod_search: return multiple results for search by id
- mod_admin: improve layout of thumbnails in find dialog
- mod_admin: find dialog: visually distinguish existing connections for current predicate
- mod_admin: align connection dialogs to browser top
- mod_admin: remove redundant intro text
- mod_admin: find dialog: visually distinguish existing connections for current predicate
- mod_base: safer defaults
- mod_base: safer defaults: add type "submit" to submitting buttons
- mod_base: add bootstrap classname to invalid form
- mod_editor_tinymce: link to correct codemirror
- mod_editor_tinymce: use bootstrap classnames

- mod_admin: admin_connect_select: fix actions in cases the value undefined is passed from a template
- docs: refactor custom admin edit page
- revert: template comment not allowed
- mod_admin: remove margin from empty tree list
- Revert "mod_base: safer defaults"
- Revert "mod_base: safer defaults: add type "submit" to submitting buttons"
- doc: describe "s" formatting character
- mod_menu: allow nested divs inside the list item
- mod_admin: make labels in list items visible
- mod_admin: prevent emptied tree list taking up space
- mod_menu: position all dialogs at the top
- mod_admin: show default text when no features are enabled
- zotonic_status: fix error message
- doc: update zotonic_status site screenshots
- mod_admin: tweak colors
- mod_admin: tweak panel heading
- mod_admin: align variables with bootstrap config
- mod_admin: optimize layout of date fields
- mod_admin: tweak colors
- mod_admin: make growl messages less dark
- mod_editor_tinymce: update codemirror plugin
- mod_editor_tinymce: replace codemirror plugin with default code editor

Maas-Maarten Zeeman (3):

- core: Fix for using template rendered file requests
- core: Fix for uncollapsing lib files without a dirname
- mod_component: Refactor of component injection code

Marc Worrell (55):

- mod_email_status: styling of some buttons.
- core: better error returns for m_identity:set_username_pw/4
- mod_backup: fix periodic backup - broke because of added 'is full backup' flag in the backup list.
- Fix a problem where the connected depictions were not always updated when adding a depiction.
- Make the admin 'replace media' buttons an anchor element instead of a button.
- Ensure that newly created connected content is in the same content group as the subject of the connection.
- core: new pivot task return: {delay, Delay, NewArgs}
- mod_filestore: changes to the 'move to local/remote' queries. This fixes an issue where moving many files at the same time results in timeouts during move initialization.
- core: refactor the template compiler, use separate compilation process so that 'changed' checks can continue whilst compiling.
- mod_acl_user_groups: add type="submit" to rule-add button
- core: fix return value check of password change.

- mod_acl_user_groups: set the default user group for user 1 and sudo to the adminstrators-group. Fixes #1042
- mod_admin: fix a problem with uploading media if no subject_id is present
- mod_acl_user_groups: fix default manager-group membership for admin or sudo
- mod_contact: add form-group class to divs, this fixes the problem that validation errors were not flagged. Thanks to @fredpook
- mod_search: add query term 'match_objects'. This is similar to the {match_objects id=...} query.
- core: if postback is handled, also return the page message queue.
- mod_base: fix fetching queued transport messages
- mod_translation: add config 'mod_translation.force_default' to prefer the default language above the browser accept languages.
- acl: add option to check permission as-if the typical member has been logged in. Issue #1050
- core: better error message in z_notifier.
- core: fix find_value error in m_acl. Fixes #1052
- mod_acl_user_groups: merge fix for sudo user groups.
- core: fix a problem where the authoritatve uri could be requested by the datamodel installer before the id dispatch rule was known.
- core: change stemming of the full text indexes.
- core: fix a problem with abs_url and urls starting with '//'. This now correctly adds the protocol.
- core: Stop email transmission if sender has been disabled or deleted. Fixes #1046
- core: defined an admin as someone who is either user 1, sudo, or allowed to use the mod_admin_config. Fixes #1033
- core: always return an error when looking up 'undefined' with name_to_id.
- mod_search: fix a problem where search texts where stemmed twice.
- mod_search: fix for dutch wildcard tsquery with dutch stemming; 'overstee' and 'oversteek' were mapped to 'overstee':* and 'overstek:*
- erlydtl: hack fix for escape filter application in arg lists. Like '{foo bar=x|escape}'. For now direct mapping to the force_escape filter.
- core: add dialogs and routines to move categories/content-groups/user-groups/predicates if one is deleted. Fixes #1041
- mod_admin: fix js problem with touch js
- mod_atom: fix test for new z_stdlib:strip
- New locked version for z_stdlib and s3filez
- mod_base: fix controller_comet for a problem where comet didn't flush on page data if a postback controller fetched the data before. (picked from d855b1abec4b55e0c0ab7c77f4a66c08da3194bd)
- mod_base: export the post-loop in controller_comet.
- mod_search: show growl error message on missing predicates and categories. Missing category matches the empty range. Fixes #998
- mod_search: ignore 'undefined' categories in search.
- mod_rest: fix a problem where a backup file could not be restored if the content group id was unknown. Fixes #1011
- mod_search: fix fulltext query sql. Fix searching for prefixes which are wildcards. Fixes #1061
- core: z_utils:is_empty/1, define St. Juttemis date as an empty value..

- mod_base: add parameter less filter 'filter' to remove empty values from a list.
- erlydtl: stricter definition of gb_trees, added lookup of m_search_result properties.
- core: addes z_pivot_rsc/get_task/1,/2,/3 and /4. Used to check pivoted tasks
- mod_admin: add filter temporary_rsc. Fixes #1044
- docs: add some doc placeholders and documentation for the action mask_progress.
- mod_video: add '-pix_fmt yuv420p' for QuickTime compatibility
- mod_email_status: don't register a 'sender_disabled' error with the recipient's status.
- core: expose pivot_location_lat, pivot_location_lng and pivot_geocode_qhash as m_rsc properties.
- core: correct identify of MS Visio files.
- core: allow 'gray' and 'grey' for the grey image filter.
- core: set `'Cache-Control-Allow-Origin: *'` on all controller_file replies.
- Lock deps. Fixes a problem where the empty url was changed into '/'. Fixes #1066

Release 0.13.6

Welcome Zotonic 0.13.6, released on December 14, 2015.

This is a maintenance release of Zotonic 0.13

Main changes are:

- Fix the deps version lock by adding USE_REBAR_LOCKED
- New bin/zotonic command to start without a shell: *start-nodaemon*
- Overview of edges in the mod_admin
- New module mod_admin_merge to merge resources
- Retained messages for ~pagesession topics
- Performance fixes for api calls
- Significant better revision history in mod_backup
- Add `no_touch` option to `m_rsc:update/4`
- Many smaller fixes

The following 0.13.7 release will incorporate some performance and stability enhancements from the master (1.0-dev) branch.

Commits since 0.13.5

There were 96 commits since release 0.13.5.

Big thanks to all people contributing to Zotonic!

Git shortlog

Arjan Scherpenisse (1):

- core: Revert edown revision bump due to compilation breakage

Arthur Clemens (24):

- Tweak input border border to match buttons

- doc: typo
- mod_admin_predicate: fine-tune display of items
- mod_signup: update translation strings, update Dutch
- mod_authentication: update translation strings; update Dutch
- doc: move number formatting to strings
- doc: update examples so that they will work
- mod_admin: style disabled tabs
- doc: fix make script
- doc: fix make script
- mod_admin: add category name to category dialog on edit page
- mod_admin: add category preselection to connect-find dialog
- filewatcher: handle spaces filepath in less command
- filewatcher: handle spaces filepath in less command
- mod_admin: reverse created/modified
- Remove borders from connections list
- mod_admin: align category param between find and new tabs
- mod_translation: handle concatenated lists
- mod_admin: generate translation strings, update NL
- mod_acl_simple_roles: typo
- mod_admin: add dependency mod_mqtt
- doc: improve docs on CORS
- mod_editor_tinymce: remove codemirror init code
- mod_admin_merge: grammar and text tweaks

David de Boer (2):

- Add modules model docs
- Fix query export

Maas-Maarten Zeeman (6):

- mod_mqtt: Added a way to pass js arguments for mqtt wires.
- mod_base: Use beacon api to get good page close feedback for chrome/firefox and recent android users
- mod_mqtt: Added retained messages for local (~pagesession) topics
- mod_component: Updated mithril to version 0.0.2
- mod_mqtt: Just return the topic if it is not a site topic
- core: Make sure looking up translations always succeed. Fixes #1103

Marc Worrell (57):

- mod_acl_user_groups: fix acl check for module use rights. Fixes #1071
- Merge pull request #1064 from driebit/modules-model-doc
- Add USE_REBAR_LOCKED file to use the locked config
- Fix erlware_commons version, as the erlware_commons master requires rebar3

- Copy Travis config from master
- Lock version of new mochiweb and qdate
- mod_admin_predicate: add overview pages of edges, optionally filter by predicate. Issue #1075
- core: tuning the full text search. Add option to exclude certain predicates from adding the object's title to the pivot text. Set different full text ranking weights and behaviour
- mod_admin_predicate: rename 'connection' to 'page connection'
- mod_admin: add the 'all connections' button to the dashboard
- mod_search: fix for rank_behaviour arg in simple fullext query.
- mod_base: add convenience arg to lazy scomp. If 'template' argument is defined then all arguments are wrapped in an update action.
- core: add controller_http_error for handling HTTP errors in other controllers.
- Locked new webzmachine (needed for the error controller)
- mod_survey: add option to send answers to the respondent.
- mod_survey: change some texts to make options clearer.
- filewatcher: use os:find_exectutable/1 instead of os:cmd("which ..") to find command line executables. Fixes #1078
- core: check for the presence of 'identify', 'convert' and 'exif' before execution. This gives better error messages. Issue #1078
- core: fix 'undefined' warning if rsc is inserted with content_group_id set to 'undefined'.
- mod_admin_identity: let all people with use.mod_admin_identity manage usernames and passwords. Fixes #1077
- mod_acl_user_groups: refine 'block' semantics, now blocks fo the mentioned user group.
- Undo changes by commit c10055bdd581877d9df113407d8083877a1bc6b6
- erlydtl: change generated (image tag) code to never echo errors inline in the template. Instead call lager or error_logger.
- core: fix tag/viewer generation (typo)
- core: fix a problem in z_email_server with mailing attachments by rsc id.
- mod_base_site: add z.clickable.js
- mod_base_site: add email to the share page dialog.
- mod_admin: change 'replace media item' text to 'add media item' if no medium is present. Thanks to @fredpook.
- mod_admin: use live templates for the connection lists.
- core: fix order of subcategories returned by is_a
- core: cleanup text to pivot and search. Remove '-', '/', and tags. Unescape the remaining text.
- smtp: mark emails from the mailer-daemon as automatic emails.
- Merge changes by @CyBeRoni issue #1082
- mod_search: allow to check on 'is null' and 'is not null' using '='/'<>' and 'undefined'
- New z_stdlib.
- Lock parse_trans to newer version.
- Move to uwiger/jobs instead of esl/jobs
- New z_stdlib

- mod_admin: set 'inputmode' attributes. Remove 'type=url' to prevent Chrome from checking the input. Issue #1093
- mod_admin: allow changing edges on the 'linkable' permission instead of 'editable'. Fixes #1098
- mod_base: fix some performance regressions in controller_api and z_service. Add simple Techempower json benchmark.
- New webzmachine - fixes logging of 500 errors.
- mod_backup: redo user interface of revision history
- Merge pull request #1088 from driebit/fix-export-query
- core: add api to return edge properties. Add 'no_touch' option to m_rsc_update, and option to set creator_id and craeted on edge. Fixes #1087
- mod_acl_user_groups: add a 'published' check to SQL searches for non-admin users. Small cleanup in z_search. Fixes #1081
- core: remove sql errors when enabling modules in site (re)start.
- core: log errors returned from collecting the custom pivots.
- core: add z_pivot_rsc:pivot_delay/1, makes it possible to delay pivoting when performing many resource updates.
- core: trim pivot fields like name and city.
- core: new functionality to merge two resources. Adds module mod_admin_merge and #rsc_merge{} notification.
- docs: new placeholders for mod_admin_merge and some new controllers.
- core: fix rsc_gone lookup for new location
- core: when merging, also merge properties (todo: map languages). Fix a problem in the admin new-rsc dialog when the category selection is set to '*'
- mod_admin(_merge): prevent form submit when pressing enter in the connect or merge search fields.
- mod_content_groups/mod_admin_category/mod_acl_user_groups: Delay pivoting whilst peforming updates of many resources.
- Prepare for 0.13.6 release, preliminary release notes.

Marco Wessel (5):

- Allow properly starting up in the foreground
- Script should be executable of course.
- Reinstate heart
- Reverse logic
- Be correct about quoting variables

loetie (1):

- Typo inside is_allowed check mod_import_csv.erl

Release 0.13.7

Welcome Zotonic 0.13.7, released on 1 February, 2016.

This is a maintenance release of Zotonic 0.13.

Main changes are:

- This release is the first in the new monthly release schedule.

- From now on, release tag names no longer have the `release-` prefix.
- Combine `if` and `with` tags in one expression.
- Improved performance through dispatch compiler.
- New search options: `unfinished`, `unfinished_or_nodate`.
- New image option `lossless=auto` that prevents images from being converted to JPG.

Commits since 0.13.6

There were 60 commits since release 0.13.6.

Big thanks to all people contributing to Zotonic!

Git shortlog

Arjan Scherpenisse (2):

- z_media_preview: Add lossless=auto option to keep images as PNG after resizing
- z_media_preview: Remove debug statement

Arthur Clemens (4):

- mod_admin_merge: improve error feedback
- mod_admin_merge: show error message when trying to delete the admin page
- mod_admin_merge: more typo fixes
- mod_admin_merge: prepare interface for “merge only”

David de Boer (6):

- Fix category preselection
- Fix template not found error
- Fix e-mail address
- Add cookbook entry for new user’s group
- Merge pull request #1153 from DorienD/formerror
- Prepare release 0.13.7

Dirk Geurs (1):

- mod_base: Issue #1084: Force model queries into values more often

Dorien (5):

- [survey] added bootstrap 3 class support, added styles for narrative question
- Added config key for content group when user registers
- Added documentation for mod_signup content group config
- changed default content group to undefined
- removed has-error class from form

Maas-Maarten Zeeman (1):

- core: Removed usage of md5_mac and sha_mac.

Marc Worrell (39):

- Merge pull request #1108 from DorienD/fix-bootstrap-survey

- Merge pull request #1105 from driebit/fix-category-preselection
- mod_video: fix preview generation.
- docs: fix generation for modules stubs. Fixes #1091
- mod_base: fix reference for (in)validFormClass. Issue #1107
- mod_search: add option to define a filter term with 'or' query
- Merge pull request #1112 from driebit/fix-template-not-found
- mod_base: fix ubf.encoce typo. Thanks to Werner Buchert
- core: fix crash in rsc duplicate.
- mod_admin: show unique name in overview, slight cleanup of overview loop code.
- core: fix a problem where a revert sql query could have twice the creator_id.
- core: fix a problem where old file contents would be served for gzip-encodings.
- mod_admin: fix category pre-selection for insert and connect dialog.
- core: better fallback for unexpected file 'identify' results.
- erlydtl: add optional 'as var' to the if/elif expressions. Fixes #1085
- docs: add 'if-with' documentation. Issue #1085
- erlydtl: add missing to_list/1 function from master.
- erlydtl: merge .yrl file from master.
- core: use dispatch compiler instead of dispatch server. Minor adaptations from the version in master. Issue #1121
- core: better error message on depickle problems.
- core/mod_development: fixes for dispatch-rewrites and dispatch tracing.
- mod_development: also accept path-tokens that are strings.
- mod_search: add 'edge.created (and friends) sort order option
- mod_search: if ordering by edge seq then add sub-order by id, so that the order is similar to the one used by the edge model
- m_search: log errors on invalid queries, don't crash.
- m_search: fix empty search result for erroring queries.
- core: fix a problem where the sign_key for zotonic_status could change between requests.
- mod_search: make the ranking behaviour configurable.
- mod_acl_user_groups: relax adding users to a user group. Fixes #1125
- mod_acl_user_groups: better check for 'hasusergroup' edge insert permission.
- mod_search: add 'unfinished' and 'unfinished_or_nodate' query terms.
- mod_query: let 'unfinished_or_nodate' check the date_start for nodate, as often the end date is not set.
- mod_admin_predicate: move predicates in batches of 100, take care of possible duplicate edges.
- mod_translation: set the session language if the language is changed. Fixes #1155
- mod_survey: replace some old bs2 size classes with bs3 classes.
- core: fix for z_email_receive:get_host/1
- core: fix return value for z_sites_dispatcher:get_host_for_domain/1
- mod_admin_identity: make the error-target element optional for identity-add events.
- docs: fix header for sending email

Paul Guyot (1):

- Fix issue with ffprobe returning unicode text

Witeman Zheng (1):

- add a encoder fun for the external mqtt client

Release 0.13.8

Welcome to Zotonic 0.13.8, released on 29 February, 2016.

This is a hotfix release of Zotonic 0.13 in which a leap year bug was fixed:

- Fixed bug with leap year.

Git shortlog

Lil' Dash (1):

- Fix the leap year error(issue : #1202)

Release 0.14.0

Welcome to Zotonic 0.14.0, released on 7 March, 2016.

Main changes are:

- From now on, we follow Semantic Versioning by increasing the minor instead of the patch number for our monthly releases.
- Added data model notification.
- Added managed ACL rules.
- Added `zotonic wait` and `zotonic rpc` commands.
- Added new user category configuration parameter.
- Fixed redirect to login when user has no permissions by showing 403 page with login form.
- Fixed several import and export bugs.
- Fixed Dutch translations for mod_acl_user_groups.
- Fixed status page login button.
- Improved documentation.
- Improved m.req model.

Commits since 0.13.8

There were 65 commits since release 0.13.8.

Big thanks to all people contributing to Zotonic!

Git shortlog

David de Boer (17):

- mod_import_csv: Fix CSV import for CR files
- Add release preparation script

- Restructure documentation into guides and reference
- Add data model notification
- Increase resource URI length
- Move 0.13.7 release notes to new dir
- doc: Document how to build docs
- mod_admin_identity: Add config param for new user category
- Support managed ACL rules
- mod_acl_user_groups: Improve wording and add missing Dutch translations
- Prepare hotfix release 0.13.8
- doc: Improve access control chapter
- Move release notes to proper location
- Prepare release 0.14.0
- doc: Fix link
- doc: fix notification reference
- Add 0.14.0 release notes

Lil' Dash (1):

- Fix the leap year error(issue : #1202)

Maas-Maarten Zeeman (15):

- core: Added reference to active comet connection. Preparation for connection test page
- mod_base: Added connection test page
- core: Return 0 bytes when there is nothing to transport. Related to #1097
- core: Typo in refactored controller_comet
- mod_base: removed debug statement from comet controller
- admin: Make it possible to brand the admin console
- mod_base: Removed delegate macro, it was tripping edoc generation
- core: Fixes setting and resetting ws transport timeouts. Related to #1116
- mod_base: Let postback and comet return text/x-ubf mimetype.
- build: Make it possible to separate user beams
- build: Add user_ebin_dir to zotonic.config.in
- build: Fix for the case when there is no user work
- doc: Repair underline warning.
- mod_mqtt: Restart middleman subscriber processes. Fixes #1201
- mod_ssl: Create and use a file with dh parameters. Fixes #1198

Marc Worrell (31):

- mod_export: encode YMD only date as UTC, add fallback for other dates to handle illegal dates.
- mod_export: suppress ST_JUTTEMIS and year 9999 dates in the output.
- Lock new depcache. Fixes a problem where the wrong depcache could be used for lookups. Fixes #1172
- core: refactor transport of multipart posts and comet polls.

- core: fix specs of refactored z_parse_multipart functions.
- mod_base: only initialize callback forms after javascript delegate calls, not after other delegates. Issue #1159
- core: allow 'none' as crop argument.
- core: allow flexible 'visible_for' settings.
- docs: add doc for m.category.text.tree_flat
- core: use the 'image' dispatch rule to generate image urls.
- core: add more information and better docs for the m.req model.
- core: (m_rsc) allow template access to page_url and other 'public' properties of non accessible resources.
- core: show 403 page with option to login instead of redirecting to login controller. Fixes #1148
- core: fix 'provide_empty' in http error controller.
- core: fix erlydtl debug for binary filenames.
- mod_twitter: drop tweet id 0
- mod_translation: add interface to change the i18n.language_stemmer configuration
- mod_oembed/mod_video_embed: add the medium properties video_embed_service and video_embed_id. Fixes #941
- mod_base: added actions overlay_open/overlay_close
- mod_backup: fix acl check for revisions
- core: fix redirect returns within dispatcher for alternate hosts.
- mod_acl_user_groups: fix a problem with edge checks for hasusergroup edges. Fixes #1199
- mod_authentication: fix ru translation for 'Sign in' Fixes #1197
- core: allow '{lossless, false}' in mediaclass definitions.
- core: fixes for initialization and startup of new sites.
- mod_acl_user_groups: fix schema initialization. Fixes #1200
- core: fix for check of db schema exists.
- core: better check query for schema existance.
- core: filter the watch_dirs on dead links, this fixes a problem with inotify stopping on removed link targets
- core: fix a problem where the comet loop can crash if the page process dies.
- mod_survey: do not crash on unknown survey question types when preparing charts.

Marco Wessel (1):

- scripts: Add zotonic-rpc and zotonic-wait commands

Release 0.15.0

Welcome to Zotonic 0.15.0, released on 4 April, 2016.

Main changes are:

- Added `is_number` template filter.
- Added `persistent_set` template action. resource.
- Added mod_media_exif.

- Improved mod_export.
- Upgraded TinyMCE to 4.3.7.
- Fixed #1216 by changing `lossless` media option to `false` (instead of `auto`).
- Fixed #1148 by adding 403 page for logged in users that have no access to the current
- Fixed #1205: DPI for resized images.
- Fixed #1224: limit access to mod_rest and mod_export to users that have `use` permission.
- Fixed #1221: order of Growl messages.

Commits since 0.14.0

There were 60 commits since release 0.14.0.

Big thanks to all people contributing to Zotonic!

Git shortlog

Arthur Clemens (3):

- mod_admin: remove extraneous closing div
- mod_admin: move footer to its own row
- mod_admin: add a bit of space to the footer

David de Boer (4):

- mod_admin: Fix link to search docs (#1220)
- doc: Fix ACL rule fixture syntax
- scripts: Fix comment
- Prepare 0.15.0

Marc Worrell (53):

- core: fix a problem where the comet loop can crash if the page process dies.
- mod_base: add 'is_number' filter.
- mod_backup: move the list of known resource properties to m_rsc.
- mod_export: refactor export, separate encoder functions. Added xlsx export encoder.
- mod_survey: remove controller_survey_results, mod_export is now used.
- docs: remove controller_survey_result docs.
- docs: add documentation for filter is_number
- mod_export: add vcalendar export (ics)
- mod_export: connect mod_export to the 303 handling of controller_id
- mod_export: remove #export_resource_data observe, this is already handled in export_encoder:do_body/2
- mod_base: new action 'persistent_set'
- docs: fix inline quote error.
- docs: add persistent_set to the doc tree
- core: also look into ~/.zotonic/<major-version>/.. directory for configuration.

- mod_export: filter tags and unescape html with spreadsheet exports. Add 'raw' option to not filter/unescape.
- mod_export: fix double reverse of xlsx rows
- mod_survey: also show the prompts in the answers exports.
- core: use Erlang 'exif' module and add mod_media_exif to extract resource properties (gps, orientation, crop, date)
- Run travis on 0.x
- docs: add simple documenttaion for mod_media_exif
- core: add mod_media_exif to the core modules.
- core: fix for focus point calculation.
- tinymce: add version 4.3.7. Add option to add captions to the body media. Also: * Add a template for the image options dialog * Move some named wires to _editor.tpl to prevent multiple initialization for every single editable body.
- mod_editor_tinymce: remove version 4.0.26 and 4.1.6
- mod_editor_tinymce: remove the deleted tinymce versions from the configure dialog
- mod_editor_tinymce: include the correct css version
- mod_export: simplified download buttons for the admin sidebar
- mod_export: add explanation for event download
- mod_authentication: add 403 page with logon form (or redirect button for secure page)
- Switch to nlfiedler/erlang-exif.git instead of our own branch
- Switch to nlfiedler/erlang-exif.git instead of our own branch
- mod_base: remove comet streamhost from zotonic js
- core: remove mentions of streamhost (which is unsupported)
- New mochiweb
- mod_filestore: remove GreenQloud - they transferred their business to another company.
- core: start using psql 'IN (SELECT(unnest(::int[])))' instead of concatenated id strings.
- mod_twitter: fix a problem where httpc sessions were not closed.
- New twerl library
- mod_base: restart ws/comet if navigating back to the page in iOS/Safari
- mod_base: tune stream restart on pageshow of persisted pages.
- mod_base: better session check on pageshow. Still a problem if tinymce is enabled and the page is re-visited for the 2nd time (twice back to the page).
- mod_mailinglist: add some useful shortcuts to the edit sidebar panel
- New tw* erl dep
- core: ensure that resized images have a density of 72DPI. Fixes #1205
- Fix media preview test for dpi forcing
- mod_base: show newer growl messages on top. Fixes #1221
- core: change media preview option 'lossless' default to 'false' (instead of 'auto'). Fixes #1216
- Fix a problem with filtering on content-group in searches.
- mod_acl_user_groups: fix a problem where the ACL tree expand could not find some entries.

- mod_acl_user_groups: fix problem adding new rules. Stabilize the order of rules by including the rule creation date and id into the sort order Split system content groups in pull-down, to clearify that 'all' doesn't apply to the system content groups.
- mod_export: limit exports to users with mod_export.use permission. Refactor export api, simple privacy filter for email address. Issue #1224
- mod_rest: add acl check for mod_rest.use. Issue #1224
- mod_logging: fix a problem with filtering on content-id and other-id.

Release 0.16.0

Welcome to Zotonic 0.16.0, released on 2 May, 2016.

Main changes are:

- Fixed #1099: added collaboration groups.
- #1227: Added ip_whitelist configuration option to block unwanted access when admin password is 'admin'.
- Serve status site with 503.
- Fixed #1229: users cannot update their person resource.
- Fixed #1236 by removing is_authoritative access check.
- Fixed #1245 by improving the 403 page.
- Fixed #1147 by passing `qargs` to search.
- Fixed #1230: Firefox file upload error.
- Fixed #1248: Dutch month abbreviation.
- Fixed #1250: view button in case of no update permissions.

Commits since 0.15.0

There were 39 commits since release 0.15.0.

Big thanks to all people contributing to Zotonic!

Git shortlog

Arjan Scherpenisse (4):

- zotonic_status: Serve the root page with HTTP 503
- mod_acl_user_groups: Fix permission issue
- mod_acl_user_groups: Fix typo in edit check on the collab group itself
- scripts: Fix prepare-release.sh

David de Boer (5):

- mod_acl_user_groups: Automatically publish managed ACL rules
- doc: Add Sphinx extlinks extension
- mod_l10n: Fix #1248 Dutch month abbreviation
- doc: Update for new lossless default (6e5692d0cab0eb5a125a05a3ab02933a0e6829e4)
- Prepare release 0.16.0

Marc Worrell (29):

- Add a whitelist of IP addresses that can use the default 'admin' password to login into sites.
- Add doc for ip_whitelist config. Fix ip6 netmask
- mod_search: add search argument 'qargs'
- mod_admin: add 'object_id' as optional argument to the connect dialog. Remove template from the 'update' notification, as it is '_rsc_item.tpl' anyway.
- modules: correction for some icons and small textual changes.
- mod_admin_predicate/category: fix a problem with moving categories and predicates.
- Add 169.254.0.0/16 and fe80::/10 to the default ip_whitelist
- core: add z_module_manager:activate_await/2, z_notifier:await/2 and /3.
- core: rename z_module_manager:activate_wait to activate_await
- mod_base: fix a problem with Firefox when uploading files via the postback controller. Fixes #1230
- mod_base: fix a problem with firefox and sortable. Fixes #1239
- mod_acl_user_groups: added support for collaboration groups. Issue #1099
- Fix a problem where an user can't update their own 'person' resource. Issue #1229
- mod_acl_user_groups: fix a problem where 'sudo' was not handled properly during rsc updates. Issue #1229
- mod_acl_user_groups: remove default 'view' rights on collaboration group.
- core: do not require additional permissions to change the is_authoritative flag. Fixes #1236
- docs: remove mention of streamhost, as it is removed from the code. Fixes #1228
- mod_survey: changes for editing surveys and readable display of survey answers.
- mod_export: fixes for export.
- mod_export: fix header value lookup for {trans, _} tuples.
- mod_base: in filter temporary_rsc, don't crash on insert access error but return 'undefined'
- mod_admin: on edit page, make 'view' button into an anchor if no edit rights. Fixes #1250
- mod_acl_user_groups: do not give view rights on unpublished rsc, unless user has edit permission.
- mod_authentication/mod_base: better 403 page and translations. Fixes #1245
- mod_acl_user_groups: 'edit' should be 'update', use m_rsc version of is_published_date
- core: ensure in m_rsc that 'language' is a list of known languages.
- mod_signup: use foldr for signup_form_fields, let higher prio modules win.
- mod_acl_user_groups: tune access permissions for collaboration groups. All collab group members can view the collab group. If someone can update/link/delete a collab group, then that user can do the same on the collab group content. Rename the config collab_group_edit to collab_group_update.
- mod_acl_user_groups: members of a collaboration group can view each other.

Osei Poku (1):

- doc: Fix minor errors in documentation

Release 0.17.0

Welcome to Zotonic 0.17.0, released on 6 June, 2016.

Main changes are:

- Added #1274: SNI support on Erlang 18.3 and higher.

- Added #1284: default ACL rules.
- Added #1240, #1276 and #1283: documentation for ACL user groups, task queue and Google Analytics.
- Added #1265 and #1268: sanitise SVG uploads and link tags.
- Fixed #1285: be less verbose when inserting ACL rules.
- Fixed #1272: Vimeo embeds.
- Fixed #1271: protected media security.
- Fixed #1207 by giving `m_rsc.uri` property precedence over generated URI.
- Fixed #1132 by setting `Content-Security-Policy` header.

Commits since 0.16.0

There were 54 commits since release 0.16.0.

Big thanks to all people contributing to Zotonic!

Git shortlog

Arjan Scherpenisse (11):

- scripts: Fix prepare-release.sh
- doc: Fix version nr; add missing rst files; update installation requirements
- doc: Update sidebar, update installation instructions
- admin: Show collaboration groups in content dropdown on overview
- mod_admin: Show content groups + page size filters on media overview
- zotonic_status: Show site name next to URL
- mod_acl_user_groups: Fix typo in edit check on the collab group itself
- mod_email_dkim: Add DKIM signing of outgoing emails
- mod_base: Add 'without' filter
- doc: Add filter_without to filter toc
- mod_acl_user_groups: Fix permission check for adding members/managers to group

Arthur Clemens (1):

- core: use short notation to include the header

David de Boer (14):

- deps: Lock erlang_localtime
- doc: Fix absolute/relative URL terminology
- mod_admin_identity: Fix verification e-mail URL
- doc: Update release branch name
- Add 0.16.0 release notes
- Clean up README and fix dead links
- doc: Document Google Analytics
- doc: Document media caption
- base: Fix figcaption tag

- mod_acl_user_groups: Insert default ACL rules (see #1131)
- doc: Document the task queue
- mod_acl_user_groups: Be less verbose when editing and publishing ACL rules
- doc: Document mod_acl_user_groups
- doc: Fix typos

Maas-Maarten Zeeman (2):

- build: Locked new mochiweb in order to support SSL on IE9 and 10 on OTP 18+
- core: Move ssl listeners to the core and support SNI.

Marc Worrell (26):

- mod_signup: use foldr for signup_form_fields, let higher prio modules win.
- deps: switch to original erlang_localtime from dmitryme. Issue #1036
- mod_acl_user_groups: tune access permissions for collaboration groups. All collab group members can view the collab group. If someone can update/link/delete a collab group, then that user can do the same on the collab group content. Rename the config collab_group_edit to collab_group_update.
- mod_acl_user_groups: members of a collaboration group can view each other.
- core: add support for '–' operator, extend support for '++' operator.
- core: add sanitization on the contents of uploaded SVG files. Issue #1265
- Lock new dispatch_compiler. Fixes #1261
- core: in m_rsc, always let the uri property take presendence above the local 'id' rule for non-informatonal uri generation. Fixes #1207
- core: fix a problem where a file could be downloaded iff the file is not stored via a filestore. Fixes #1271
- mod_video_embed: fix a problem where Vimeo embeds did not show a preview images. Fixes #1272
- mod_admin: in depiction upload dialog, enable the 'upload' tab by default.
- core: in html sanitize, add 'noopener noreferrer' to <a/> tags with a 'target' attribute.
- mod_video_embed: better handling of 404 when fetchin Vimeo thumbnail.
- core: added extra SVG sanitization.
- Lock new z_stdlib for SVG sanitization. Issue #1265
- mod_survey: also mail all uploaded files to the 'survey_email' email address.
- mod_admin: set X-Frame-Options: SAMEORIGIN header for admin pages. Issue #1132
- mod_admin: fix is_authorized/2 in controller_admin_edit
- mod_base: set CSP sandbox header if an user uploaded file is served with controller_file. Issue #1132
- mod_acl_user_group: add option to move resources to collaboration groups the user is not member of.
- mod_admin: allow to select multiple connection in the connection-find dialog. Use the option 'auto-close' to change the text of the close button to 'cancel'.
- mod_acl_user_groups: better overview of rules. Use dialog for editing rules.
- mod_acl_user_groups: in acl rule edit, clear collaboration group search when group is selected.
- mod_acl_user_groups: fix filtering on content groups when searching. This fixes a problem when an user is allowed to see all or specific collaboration groups via the ACL rules.
- mod_survey: fix layout of admin options.

- mod_survey: in the emails, also show any 'injected' fields. This allows the template to dynamically inject some answers without corresponding questions.*

Release 0.18.0

Welcome to Zotonic 0.18.0, released on 4 July, 2016.

Main changes are:

- Removed module mod_acl_adminonly from core (#1289). Please use *mod_acl_user_groups* (page 271) instead. If you need mod_acl_adminonly, the module is available in a separate repository.
- Added compatibility with OPT 19 (#1323).
- Added page blocks to the full text search index (issue:*1131*).
- Added support for Google Universal Analytics (#1281).
- Added testsandboxdb site for integration tests against a database (#1235).
- Fixed multiple images captions (#1311).
- Fixed hierarchy tree not updating after edit (#1293 and #1309).
- Fixed inserted video in TinyMCE not shown (#1304).
- Fixed anonymous user editing.

Commits since 0.17.0

There were 33 commits since release 0.17.0.

Big thanks to all people contributing to Zotonic!

Git shortlog

Arthur Clemens (4): mod_mailinglist: fix unsubscribe url syntax mod_admin: add param cat_exclude to overview mod_admin_identity: use same column headers as overview mod_admin: tweak Dutch translation

David de Boer (3): tests: Add testsandboxdb site core: Fix inotify statup log message tests: Make build fail if Zotonic fails to start

Maas-Maarten Zeeman (2): mod_ssl: Redirect http and https to the outside ports mod_base: Use utf-8 encoding for application/javascript files

Marc Worrell (22): core: fix z_module_manager:activate_await/2. Also fix a problem where upgrading a module could activate that module. Issue 1235 core: only start testsandboxdb during tests. mod_base: more generic escapejs filter. Issue #1281 mod_admin: fix is_session_alive status, thanks to @Dirklectisch core: let the ACL decide of anonymous users can edit something. Thanks to @Dirklectisch Locked new depcache. mod_editor_tinymce: fix a problem where setting the options of one media item changed the options of all media item. Fixes #1311 mod_video_embed: fix an issue where inserting a video inside tinymce doesn't show the video. Fixes #1304 mod_admin: prevent iframe overflowing from media preview on edit page. mod_admin: fix an issue where the menu editor was not updated after creating a new resource. Fixes #1309 Fixes #1293 mod_admin: in connect dialog, fix enabling upload tab without depiction predicate. core: also pivot texts in blocks. Fixes #1113 core: only install the skeleton modules on initial site install. Fixes #1279 core: remove debug from pivot docs: remove 'resource' from the contactform cookbook. Fixes #1166 core: fix return value of z_install:install_skeleton_modules/1 core: pivot block texts to priority B not A. mod_base_site: misc fixes for base site. mod_survey: set the language of result emails to the default system language. Fixes #1324 core: in activate_await, check the module process pid, and not the db. Fixes #1325 core: fix whereis check in activate_await. Issue #1325 core: OTP-19 compatibility changes for 0.x (merging from master). Issue #1323 (#1327)

Péter Gömöri (1): Fix xref issues (#1315)

Tah Teche (1): mod_seo: Switch to Google Universal Analytics

Release 0.19.0

Welcome to Zotonic 0.19.0, released on 1 August, 2016.

Main changes are:

- Removed module mod_acl_simple_roles from core (#1328). Please use *mod_acl_user_groups* (page 271) instead. If you need mod_acl_simple_roles, the module is available in a separate repository.
- Added site tests (#1331).
- Added View dropdown menu in admin (#1345).
- Added possibility to return JSON error objects from API service controllers.
- Added `m.req.is_bot` (#1340).

Commits since 0.18.0

There were 22 commits since release 0.18.0.

Big thanks to all people contributing to Zotonic!

Git shortlog

Arjan Scherpenisse (9): Implement running site-specific tests (#1332) core: Add z_sitetest:{watch,unwatch} to speed up test driven development API: Allow process_{post,get} to throw() error for shortcircuiting core: Only run sitetests when compilation succeeds mod_acl_user_groups: Documentation fixes; change 'edit' -> 'update' API: Fix logic bug in try/catch clause handling for process_{get,post} doc: Add note that testname cannot containing underscores API: Log crashes, serve HTTP 500 on uncaught exception filewatcher: reload module first when running all sitetests for a site

Arthur Clemens (3): docs: Describe error handling from API service Fix include_lib mod_base: controller_api: add option to pass JSON error object

David de Boer (4): core: Move mod_acl_simple_roles out of core into separate repo (#1328) doc: Add .pot generation function call admin: Add view dropdown menu (#1345) doc: Fix build

Marc Worrell (6): core: add forward-compatible z_utils:name_for_site/2. Issue #1333 mod_admin: add possibility to disconnect connections via the connection-dialog Fixes #1339 core: add 'is_bot' property for m.req Fixes #1340 core: use the ua_classifier supplied is_crawler flag. Fallback to the hardcoded list for unknown user agents (example: curl). Fixes #1340 core: in z_user_agent allow WebSocket and websocket. z_user_agent: make the upgrade comparison case insensitive

Release 0.20.0

Welcome to Zotonic 0.20.0, released on 5 September, 2016.

Main changes are:

- Added an MQTT notification when identity is deleted (#1376).
- Fixed menu separators in the admin menu (#1388).
- Fixed some tests not being run (#1392).
- Fixed memory leak in Comet en WebSocket connection handling.
- Renamed 'App Keys and Authentication Services' admin menu item to 'External services' (issue:*1389*).

- In the admin, moved CSV import to the Content admin menu and require only use mod_import_csv permissions (#1350).
- Removed Perl dependency (#1373).
- In many places, the resource's unique name and id can now be used interchangeably (#1356).
- Several documentation improvements, including an example configuration for SSL termination using Nginx (#1359).

Commits since 0.19.0

There were 36 commits since release 0.19.0.

Big thanks to all people contributing to Zotonic!

Git shortlog

Arjan Scherpenisse (3):

- mod_acl_user_groups: Fix typo in ACL import
- core: Allow retrieving site config values from the OS environment
- mod_base: Log the stacktrace when an API method throws an error

Arthur Clemens (6):

- docs: Describe uploading files with API services
- dialog: reposition one tick after initialization
- dialog: make repositioning function accessible outside
- dialog: fix dialogs that do not have option 'center'
- dialog: add function to scroll to position
- Add style to LESS instead of CSS

David de Boer (20):

- doc: Fix build
- docker: Create lite Zotonic container (#1310)
- Add 0.19.0 release notes (#1358)
- doc: Fix typo
- doc: Document page template names (#1364)
- admin: Fix inline links (#1366)
- Add .editorconfig
- Remote trailing whitespaces from .erl and .tpl files
- core: Add MQTT notification for delete_by_type_and_key (#1376)
- mod_base: Fix Content-Security-Policy for inline PDFs (#1379)
- doc: Fix indent
- doc: Improve search docs (#1369)
- doc: Add a note that only unprotected Tweets will be imported (#1387)
- mod_import_csv: Fix untranslated cancel button
- core: For every resource id argument, allow resource name as well (#1356)

- mod_authentication: Rename menu to 'External Services' (#1389)
- mod_import_csv: Move CSV import to Content menu (#1391)
- mod_admin: Fix menu separators not shown (#1388)
- tests: Include db_tests in testrun (#1393)
- core: Remove rid from maybe_allowed (#1400)

Maas-Maarten Zeeman (1):

- core: Fix memory leak in comet and ws connection handling.

Marc Worrell (5):

- mod_menu: fix a problem where the conversion of old config-menu failed due to missing main_menu resource.
- core: allow m_rsc p_no_acl fetch of exists and id
- mod_survey: fix fetch of label names for truefalse questions.
- smtp: ensure that the logged error reason is a binary. This fixes a situation where the value {error, {error, timeout}} was returned from gen_smtp_client:send_blocking/2.
- smtp: ensure that the logged error reason is a binary. This fixes a situation where the value {error, {error, timeout}} was returned from gen_smtp_client:send_blocking/2.

Marco Wessel (1):

- Remove perl dependency (#1373)

Release 0.21.0

Welcome to Zotonic 0.21.0, released on 3 October, 2016.

Main changes are:

- Added an `acl_is_owner` notification (#1404).
- Added a Docker image for Zotonic development (#1425).
- Improved Docker images size by switching to Alpine (#1374).
- Improved password hashing by switching to bcrypt (#1390).
- Improved `is_visible` filter to support fulltext search results.
- Moved documentation to Read the Docs (#1454).
- Fixed removed observers not being detached (#1441).

Commits since 0.20.0

There were 31 commits since release 0.20.0.

Big thanks to all people contributing to Zotonic!

Git shortlog

Arjan Scherpenisse (1):

- core: Add default config for 'setup' application

David de Boer (12):

- core: Remove rid from maybe_allowed (#1400)

- tests: Fix m_identity_tests on 0.x (#1415)
- Add 0.20.0 release notes (#1423)
- mod_admin: Fix spelling error
- doc: Add deeplink to custompivot
- doc: Fix ref
- docker: Negate paths in .dockerignore to make image even smaller (#1426)
- docker: Fix compilation by switching bcrypt dep (#1434)
- doc: Improve site creation (fix #1440)
- docker: Add dev Docker image (#1425)
- docker: Fix duplicate config files (#1442)
- doc: Build 0.x docs on Read the Docs (#1454)

Maas-Maarten Zeeman (1):

- core: Use bcrypt and erlpass for password hashes (#1390)

Marc Worrell (16):

- mod_acl_user_groups: make the is_owner check a notification (#1404)
- mod_acl_user_groups: fix is_owner check.
- mod_admin: pass args to connect dialog items. Add 'thumbnail-linkable' class.
- mod_admin: in connect-dialog search result, add the class 'unpublished' for unpublished items.
- mod_admin: filter connect-list on visibility
- mod_base: let is_visible filter also accept a list of tuples {Id,Score} as returned by the fulltext search
- core: add smtp headers from the #email message.
- mod_video: fix replacement of temp re-render placeholder.
- i18n: manual merge of #1437
- mod_oembed: don't embed data for 'link' type (our metadata sniffer is better at this)
- mod_email_status: export clear_status/2
- core: fix eacces/eaccess mixup
- core: fix a problem where removing an exported observe function did not detach the observer. Fixes #1441
- core: less restrictive receive for module shutdown.
- core: skip identify errors when extracting mime format.
- mod_export: catch errors in xlsx export of illegal dates

Marco Wessel (1):

- docker: Use Alpine Linux for the Docker images (#1374)

Release 0.22.0

Welcome to Zotonic 0.22.0, released on 7 November, 2016.

Main changes are:

- Added support for managed collaboration rules (#1492).
- Added deletion interval for files stored on S3 (#1493).

- Fixed database connection recovery after PostgreSQL restart (#465).
- Fixed '53300 too many connections' PostgreSQL error (#1469).
- Fixed 'Add connection' button invisible for link permission (#1476).
- Fixed S3 delete requests being indefinitely repeated (#1488).

Commits since 0.21.0

There were 16 commits since release 0.21.0.

Big thanks to all people contributing to Zotonic!

Git shortlog

David de Boer (8):

- doc: Add 0.21.0 release notes (#1461)
- docker: Fix log_dir sed selector
- core: Fix bcrypt dep on 0.x (#1465)
- mod_acl_user_groups: Add managed collab rules support (#1492)
- mod_filestore: Fix #1231 by adding ACL explanation
- mod_filestore: Add delete interval (#1493)
- mod_acl_user_groups: Fix #1505 erroneous foreign key
- Add build deps to environment variable

Dirk Geurs (1):

- mod_admin: Show add connection buttons if linkable (#1477)

Finn Kruyning (1):

- Remove TD from admin dashboard

Maas-Maarten Zeeman (2):

- mod_export: Fix, follow export_resource_visible spec (zotonic_notifications.hrl:847)
- mod_component: Added the possibility to unmount a component

Marc Worrell (4):

- core: sanitize Microsoft Office comments, classes and styles. Fixes #1464
- core: remove trim_all option from binary:split/3 as it is incompatible with OTP 17
- core: remove transport log messages about unknown page sessions.
- Close open db connections on a 'too may connections' error. (#1470)

Release 0.23.0

Welcome to Zotonic 0.23.0, released on 5 December, 2016.

Main changes are:

- Added delete interval (and completely disabling deleting) to files stored on S3 (#1493).
- Added search rank weight configuration parameter (#1538).
- Added editing of embedded collections in admin (#1520).

- Added support for custom Google Analytics parameters (#1518).
- Fixed consistency of log messages and log metadata (#1510).
- Fixed ACL checks in admin (#1514).
- Fixed embedding not allowed for non-admins (#1545).
- Fixed initialization error in mod_acl_user_groups (#1505).
- Fixed initial tab on connect modal (#1497).

Commits since 0.22.0

There were since 22 commits since release 0.22.0.

Big thanks to all people contributing to Zotonic!

Git shortlog

Arjan Scherpenisse (1):

- scripts: Add -noshell to sitetest command

David de Boer (14):

- mod_filestore: Add delete interval (#1493)
- mod_acl_user_groups: Fix #1505 erroneous foreign key
- Add build deps to environment variable
- Prepare release 0.22.0 (#1503)
- mod_filestore: Upgrade version to add deleted column (#1506)
- admin: Switch position of translation and user menu (fix #1354)
- admin: Fix access checks (#1514)
- mod_seo: Support custom Analytics params (#1518)
- core: Improve log messages consistency (#1510)
- docker: Switch to Alpine PostgreSQL for smaller file size
- mod_menu: Fix menu and user groups not editable (#1531)
- mod_search: Quote search weight and document options
- mod_base: Validate resource name (#1504)
- mod_filestore: Fix an issue with setting deletion to 'never' (#1549)

Fred Pook (3):

- admin: Don't restrict initial tab for zmedia connect dialog (#1497)
- mod_search - Allow configuration of rank weight
- mod_editor_tinymce: translation for middle size changed to medium size (#1539)

Marc Worrell (4):

- Close open db connections on a 'too may connections' error. (#1470)
- mod_admin: edit collection in block for embedded collections (#1520)
- core: fix a problem with sanitizing classes in html. (#1526)
- Fix a problem where a non-admin could not add embedded media. Fixes #1545

Release 0.24.0

Welcome to Zotonic 0.24.0, released on 2 January, 2017.

Main changes are:

- Fixed deletion date 'never' in mod_filestore (#1549).
- Fixed handling of illegal Exif data (#1557).
- Fixed adding embedded media (#1545).
- Fixed live validation message_after error when id is invalid (#1543).

Commits since 0.23.0

There were since 16 commits since release 0.23.0.

Big thanks to all people contributing to Zotonic!

Git shortlog

David de Boer (4):

- core: Support custom search total (#1537)
- mod_filestore: Fix an issue with setting deletion to 'never' (#1549)
- Add 0.23.0 release notes
- core: Fix filezcache gzip compress pattern match

Marc Worrell (11):

- Fix a problem where a non-admin could not add embedded media. Fixes #1545
- Lock new exif (and bcrypt). Issue #1556
- mod_media_exif: fix a problem with illegal exif dates. Issue #1557
- mod_base_site: fix order of includes.
- mod_base: add UBF support for floats, proplists and dates.
- Lock new z_stdlib (for z_ubf)
- mod_base: fix a problem where a transport before init of z_pageid would reload the page.
- core: add z_mqtt:payload_user/1 function.
- core: fix a problem with identify of EPS files. Also catch crashes in Exif library on unknown or corrupt exif data.
- Lock new exif
- core: fix a problem with pre-page init pubzub subscriptions.

Álvaro Pagliari (1):

- mod_base: fix livevalidation message_after error when invalid id (#1543)

Release 0.25.0

Welcome to Zotonic 0.25.0, released on 6 February, 2017.

Main changes are:

- Added `embedded_media` filter (#1591).

- Added Atom support to mod_export (#1394).
- Added `name` search query argument (#1574).
- Fixed image crop defined in mediaclass (#919).
- Fixed adding connection to page block (#1561).
- Fixed text blocks not considered by `without_embedded_media` filter (#1566).
- Fixed unique name validation for empty names (#1579).
- Fixed filewatcher reacting to changes in log files (#1588).
- Fixed pivot index names not unique (#1598).

Commits since 0.24.0

There were since 33 commits since release 0.24.0.

Big thanks to all people contributing to Zotonic!

Git shortlog

Arthur Clemens (1):

- Add clarification in template

David de Boer (6):

- doc: Add custom controllers cookbook chapter (#1565)
- doc: Fix postback validator documentation (#1567)
- docker: Build Erlang from source (fix #1590)
- core: Raise timeout when starting site for sitetest
- doc: Fix embedded_media docs
- core: Make pivot index name unique (#1598)

Maas-Maarten Zeeman (1):

- core: Accept push queue names as binaries. Fixes #1575 (#1576)

Marc Worrell (24):

- mod_search: use publication_start as a sub sort for matching, prefer newer content.
- core: evaluate 'and'/'or' as 'andalso' and 'orelse'. Fixes #1561
- mod_base: fix a problem in show_media where utf8 characters were wrongly displayed. Fixes #1572
- mod_search: add a query term 'name' for searching on unqiue names. Fixes #1574
- mod_export: add export in application/atom+xml format. Fixes #1546 Fixes #1394
- Lock new z_stdlib
- core: let filewatcher ignore changes to log files. Fixes #1588
- core: also ignore changes to files in the mnesia directory
- mod_base: in ubf.js use Object.keys() to fetch the keys of an object. This fixes an issue where injected object.prototype functions were serialized as well.
- docs: fix build problem with 'less' code block by using 'text' instead
- mod_base: let filter without_embedded_media also consider text blocks. Add filter embedded_media. Fixes #1566

- core: rsc names with only a '_' are considered 'undefined'. Fixed #1579
- mod_base: let the unique name validation first use 'z_string:to_name/1'. Consider '_' as an invalid unique name. Issue #1579
- mod_base: allow empty unique name.
- core: use crop_center if crop is defined in the mediaclass. Fixes #919
- mod_admin: fix a problem with media preview for unknown resources.
- mod_base: fix ubf decode for plist/map types.
- mod_admin: change positioning of images in connection lists. Fixes #1570
- core: check crop_center syntax on rsc update.
- core: add email DNS Whitelisting support
- Lock new z_stdlib
- core: fix a string/binary conversion issue with the new z_stdlib
- New z_stdlib
- core: fix a problem where next/prev_day could result in a non-existant date. Fixes #1596

row-b (1):

- Minor typo in syntax fixed (#1569)

Release 0.26.0

Welcome to Zotonic 0.26.0, released on 6 March, 2017.

Main changes are:

- Added support for special characters in z-media caption (#1618).
- Added `facets` property to `#search_result{}` record (#1606).
- Fixed zmedia dialog does not open in Firefox (#1620).
- Fixed hidden dropdown menu in admin (#1603).
- Fixed a problem with UBF-encoded date values.
- Fixed forbidden response from S3 filestore triggering retries.
- Moved filezcache data and journal directories outside the dependency directory (#1601).

Commits since 0.25.0

There were 19 commits since release 0.25.0.

Big thanks to all people contributing to Zotonic!

Git shortlog

David de Boer (5):

- core: Make pivot index name unique (#1598)
- Add 0.25.0 release notes (#1600)
- core: Add facets property to search_result (#1606)
- core: Move filezcache data dir outside dep dir (#1621)

- core: Upgrade filezcache

Maas-Maarten Zeeman (1):

- core: Explicitly set the timezone of the db connection to UTC

Marc Worrell (13):

- mod_base: in ubf.js fix a problem with encoding date values
- mod_admin: fix a problem where in tree-lists the dropdown menu was hidden. Fixes #1603
- Upgrade z_stdlib.
- New z_stdlib
- mod_base: add 'is_danger' flag to the confirm dialog. This will show the 'ok' button with the 'btn-danger' style.
- Add support for special characters in the z-media caption. (#1618)
- New z_stdlib
- core: allow m_edge:get_id/4 calls with an non-existing predicate.
- mod_filestore: handle S3 forbidden as non retryable error.
- core: fix utf8 problem for R16
- mod_editor_tinymce: fix check for crop checkbox.
- core: the medium size is actually called 'middle' (for 0.x)
- mod_editor_tinymce: fix zmedia for Firefox

Release 0.27.0

Welcome to Zotonic 0.27.0, released on 3 April, 2017.

Main changes are:

- Added platform.twitter.com to whitelist (#1647).
- Added config location override based on environment variable (#1627).
- Fixed resource creation ACL check missing content group selection (#1640).

Commits since 0.26.0

There were 10 commits since release 0.26.0.

Big thanks to all people contributing to Zotonic!

Git shortlog

David de Boer (2):

- core: Upgrade filezcache
- Add 0.26.0 release notes

Maas-Maarten Zeeman (1):

- core: Add properties to acl insert to allow checks on content/collaboration groups

Marc Worrell (5):

- core: fix a problem with identifying certain m4a files.

- mod_editor_tinymce: fix zmedia for Firefox
- mod_import_csv: fetch more data analyzing the column labels.
- Facebook now returns JSON during OAuth handshake.
- Upgrade z_stdlib

Marco Wessel (1):

- Allow admin to override default config location also for rebar and such (#1627)

loetie (1):

- [core] Add platform.twitter.com to whitelist (#1647)

Release 0.28.0

Welcome to Zotonic 0.28.0, released on 1 May, 2017.

Main changes are:

- Added a temporary workaround for ErlyDTL compiler creating too many atoms (#1676).
- Added reqdata to #dispatch_host notification (#1664).
- Fixed non-admin users have no permission to upload file (#1665).
- Fixed depcache cleanup causes crash (#1671).
- Fixed uploading extremely large files crashes z_file_entry (#1650).

Commits since 0.27.0

There were 17 commits since release 0.27.0.

Big thanks to all people contributing to Zotonic!

Git shortlog

David de Boer (5):

- Add 0.27.0 release notes
- mod_seo: Rename shorturl to shortlink (#1651)
- core: Upgrade depcache
- mod_acl_user_groups: Fix non-admins denied permission to upload file (#1665)
- scripts: Raise maximum number of atoms (#1676)

Dirk Geurs (1):

- Retain reqdata for use in #dispatch_host notification (#1664)

Marc Worrell (11):

- Do not crash if resulting image is too big (#1652)
- Lock new z_stdlib
- mod_oauth: cleanup templates, add option for 'anonymous' users with only the consumer key/secret. (#1635)
- mod_video: change log level of 'Video info' lager message.
- Upgrade s3filez.

- mod_base_site: add 'showmedia' filter
- mod_base_site: fix typo.
- core: remove some throws from z_db, return error tuple instead.
- mod_survey: add es translation for 'submit'

Release 0.29.0

Welcome to Zotonic 0.29.0, released on 5 June, 2017.

Main changes are:

- Added an Exometer metric for the number of depcache evictions (#1682).
- Added Erlang maps support in ErlyDTL (#1684) and `moresults` action (#1689).
- Added ISO 8601 date support to `date` filter (#1702).
- Added a *Exometer metrics* (page 209) cookbook (#1679) and a *Server configuration* (page 63) chapter to the documentation (#1680).
- Fixed `erlang.config` not read during sitetest (#1672).
- Fixed race condition in notifications during database transaction (#1693).
- Fixed ErlyDTL compilation generating too many atoms (#1673).
- Fixed wav mimetype for Internet Explorer (#1661).

Commits since 0.28.0

There were 24 commits since release 0.28.0.

Big thanks to all people contributing to Zotonic!

Git shortlog

David de Boer (12):

- core: Add ErlyDTL maps support (#1684)
- doc: Bump year
- core: Ignore priv/filezcache
- tests: Read erlang.config in sitetest command (#1672)
- mod_base: Support maps result in moreresults filter (#1689)
- doc: Fix build (#1694)
- doc: Fix RTD theme's CSS not loaded
- core: Record depcache evictions in exometer (#1682)
- doc: Add server configuration chapter
- mod_base: Support ISO 8601 dates in filter_date (#1702)
- core: Delay notifications during database transactions (#1693)
- mod_editor_tinymce: Don't override language if already set (#1697)

Maas-Maarten Zeeman (1):

- doc: Added information on increasing nf_conntrack_buckets

Marc Worrell (9):

- core: fix a problem where compilation of templates generates random atoms. Fixes #1673 (#1674)
- mod_acl_user_groups: fix a problem with insert checks without passed props.
- core: fix a problem with .wav mime type (#1678)
- core: don't crash in rsc_gone on race conditions.
- mod_oauth: use the user-id from the app-token, not the consumer.
- Upgrade z_stdlib
- Lock new z_stdlib
- Upgrade z_stdlib
- mod_email_status: better transient error handling.

Marco Wessel (1):

- doc: Add Exometer cookbook (#1679)

yorivanloo (1):

- translation: 'bedank' to 'bedankt' (#1681)

Release 0.30.0

Welcome to Zotonic 0.30.0, released on 3 July, 2017.

Main changes are:

- Fixed a problem with escaping characters in From/Reply-To e-mail headers.
- Fixed retrieving header info when there is no wm_reqdata.

Commits since 0.29.0

There were 7 commits since release 0.29.0.

Big thanks to all people contributing to Zotonic!

Git shortlog

David de Boer (1):

- Revert "scripts: Raise maximum number of atoms (#1676)"

Maas-Maarten Zeeman (2):

- core: Fix retrieving header info when there is no wm_reqdata
- mod_mqtt: Provide a minified version of qlobber to save some kbs

Marc Worrell (4):

- smtp: fix a problem with escaping certain characaters in email from/reply-to headers.
- core: add max smtp connection per domain
- Lock z_stdlib on 0.x branch
- mod_acl_user_groups: use default content group if no content group defined.

Release 0.31.0

Welcome to Zotonic 0.31.0, released on 2 August, 2017.

Main changes are:

- Upgraded Facebook API from version 2.3 to 2.9
- Add dropdown in the admin's connect dialog to filter on content group

Commits since 0.30.0

There were 7 commits since release 0.30.0.

Big thanks to all people contributing to Zotonic!

Git shortlog

Marc Worrell (6): filewatcher: fix blacklist re for log files. mod_facebook: upgrade api to v2.9. mod_base: add optional class argument to overlay_open action. (#1757) Fix a problem with entering negative dates in the admin. Issue #1766 mod_admin: add connect-dialog filter for content group. (#1768) mod_facebook: also request first and last_name from the api.

rl-king (1): Add missing punctuation (#1772)

Release 0.32.0

Welcome to Zotonic 0.32.0, released on 4 September, 2017.

Main changes are:

- Added support for edges in CSV data (#1690).
- Sort textual admin search results on relevance (#1799).
- Fixed a problem where validation crashes on inputs with a `:` in their name (#1785).
- Fixed unintentional password changes (#1801).

Commits since 0.31.0

There were 7 commits since release 0.31.0.

Big thanks to all people contributing to Zotonic!

Git shortlog

David de Boer (3):

- core: Pass is_import to normalize (#1726)
- mod_admin_identity: Fix unintentional password changes (fix #1801)
- tests: Raise timeout to fix build failures (#1806)

Dirk Geurs (1):

- build: Install openssl in 0.x Docker container (#1792)

Marc Worrell (3):

- mod_import_csv: add support for incoming/outgoing edges in normal csv data. Issue #1690

- core: fix a problem where validation crashes on inputs with a ':' in their name (#1785)
- mod_admin: ensure that text search results are sorted on relevance (#1799)

Upgrade notes

These notes list the most important changes between Zotonic versions. Please read these notes carefully when upgrading to a new major Zotonic version.

Upgrading to Zotonic 0.12

Bootstrap CSS version 3

Zotonic has been switched over to the latest version of the Bootstrap Framework. When you are using Zotonic's `mod_bootstrap` or when you have customized the admin templates, you will need to update your templates.

A full migration guide to upgrading from Bootstrap 2.x is here: http://getbootstrap.com/migration/, a tool to help you convert your Zotonic templates is located here: https://github.com/arjan/bootstrap3-upgrader.

Upgrading to Zotonic 0.11

Global configuration changes

The global file `priv/config` has been obsoleted in place of a new global configuration file, `~/.zotonic/zotonic.config`.

To upgrade your config file, do the following:

- Make a directory in your home folder, called `~/.zotonic`.
- Copy `priv/zotonic.config.in` to `~/.zotonic/zotonic.config`
- Copy any settings from `priv/config` into the new `priv/zotonic.config` (IP addresses, etc)
- Remove the old file `priv/config`, as it is no longer in use.
- Also, move `priv/erlang.config` to `~/.zotonic/erlang.config`.

These configuration files can also be put in other places (`/etc/zotonic`, most notably), or can contain Zotonic's version number or node name when running multiple Zotonic versions side by side. See *Global configuration* (page 490) for all information on this topic.

Note: You can *not* just copy over your old `priv/config` file to the new location, as the structure of the file has changed.

Changed location of sites and external modules

The default place for user-defined sites and external modules has been changed to the defaults `user/sites` and `user/modules`, respectively.

To move your sites and modules in the right places, do the following:

- In the zotonic dir, do `mkdir -p user/{modules,sites}`
- Move any external modules: `mv priv/modules/* user/modules/`
- Move all sites except `zotonic_status` and `testsandbox` to `user/sites`.

You can change the location of the user-defined sites and modules by changing `user_sites_dir` and `user_modules` dir settings in the *Global configuration* (page 490).

Postback and javascript changes

The file `zotonic-1.0.js` now uses `lib/js/modules/ubf.js`. This file **must** be included for the Zotonic javascripts to work.

All postback, comet and websocket connection are now handled by `z_transport`. Check *Transport* (page 49) for details.

The `stream` tag has been deprecated. You can remove it from your templates. Zotonic now automatically starts a WebSocket connection on each page, unless `nostream` is given in the scomp-script tag.

Dispatch rules for files

The `controller_lib` and `controller_file_readonly` have been replaced by the `controller_file`. This controller uses the new *filestore* system in Zotonic. This enables the storage of files on remote services like S3.

If you have added your own `controller_lib` or `controller_file_readonly` dispatch rules then you have to change them to use `controller_file` instead.

The following options have been **removed**:

- media_path
- is_media_preview
- use_cache
- use of an *id* argument, use `controller_file_id` instead

See the documentation for *controller_file* (page 352) and *controller_file_id* (page 353).

Modules moved out of core

The `mod_geomap` repository has moved to its own dedicated repository. To keep using this module, you'll now need to install it as an external module: `zotonic modules install mod_geomap`. Alternatively, you can try the module `mod_geo` (`zotonic modules install mod_geomap`) , which uses Google Maps in the admin.

Database-driver changes

Due to the introduction of the new database driver, the behaviour of automatically serializing Erlang terms into the database (on `bytea` columns) has been made explicit. To enable serialization of database values, you have to tag them with the new `?DB_PROPS(...)` macro. Unserialization of terms is still done automatically.

Gotcha's

If you get this error on startup:

```
DTREE: cannot open ''
```

You can fix this by doing: `rm -rf deps/ua_classifier`, and then running `make` again.

Upgrading to Zotonic 0.10

Site config changes

The site *hostalias* option has been changed to be a list of host aliases instead of multiple pairs of hostalias attributes. Change your site's configuration from this:

```
{hostalias, "www.example.com"},
{hostalias, "www.example.net"},
{hostalias, "example.org"},
```

To this:

```
{hostalias, ["www.example.com", "www.example.net", "example.org"]},
```

Besides this change, a site's config file can now also be split into multiple files living under the `config.d/` folder within a site.

Build process

The `git` tool is now **required** to build Zotonic, even when you downloaded the release zip file. This is due to Zotonic's external dependencies now being managed with the `rebar` tool.

Misc changes

All configuration options regarding logging are now in set in the `priv/erlang.config` file, which is created by default if missing from `priv/erlang.config.in`.

Upgrading to Zotonic 0.9

CSS changes

Due to the move to Bootstrap, the following CSS changes need to be made in your templates:

Old CSS selector	New CSS selector
`.sf-menu`	`.nav`
`.sf-menu a.current`	`.nav li.active a`
`ul.pager`	`div.pagination ul`

Controllers

The Erlang modules formerly known as *Webmachine Resources* (`resources/resource_*.erl`) have been renamed to *controllers*. They live in the `controllers/` folder in a module. This was done to eliminate the confusion between webmachine resources and the "rsc" table of the Zotonic datamodel.

This means that you have to update your custom dispatch rules. Each dispatch rule which uses one of Zotonic's `resource_*` controllers, needs to be changed from this:

```
{article,      ["article", id, slug],      resource_page,      [ {template,
↪"article.tpl"} ]},
```

to this:

```
{article,      ["article", id, slug],      controller_page,      [ {template,
↪"article.tpl"} ]},
```

et cetera.

Also, when you wrote your own controllers, you need to rename your `resource_` module to use the controller prefix, and make sure it uses the new include file names.

The following include files have been renamed:

Old filename	New filename
include/resource_html.hrl	include/controller_html_helper.hrl
include/webmachine_resource.hrl	include/controller_webmachine_helper.hrl

HTTPS support

HTTPS support was moved from the core into a new module, *mod_ssl* (page 313).

The global `priv/config` options `ssl`, `ssl_certfile`, `ssl_keyfile` and `ssl_password` do no longer have an effect. See *mod_ssl* (page 313) on how to configure HTTPS support for Zotonic from 0.9 and up.

Removed controller

The under-used `resource_home` controller has been removed. Change your dispatch rules accordingly to use `controller_template`:

```
{home,  [],  resource_home,       []},
```

to this:

```
{home,  [],  controller_template, [{template, "home.tpl"}]},
```

Removed filters

The `lenght_is` filter has gone. Replace constructs like this:

```
{% if value|length_is:5 %}
```

to:

```
{% if value|length == 5 %}
```

mod_backup

mod_backup's configuration values for binary path names (*pg_dump* and *tar*) is now coming from the global `z_config` instead of the site's configuration database.

On startup you might see this message:

```
18:39:59.895 [error] z_module_manager:485 [sitename] Error starting module mod_
↪backup: {error,{missing_dependencies,[rest]}}
```

mod_backup is now dependent on mod_rest, so you should enable that module in the module manager.

mod_survey

The storage format changed slightly. For the correct display of the results of *narrative*-type questions answered before 2012-12-01, the name of the block needs to equal the name of the first narrative sub-question.

z_logger

On startup you might see this message:

```
** /home/zotonic/zotonic/deps/z_logger/ebin/z_logger.beam hides /home/zotonic/
↪zotonic/deps/webzmachine/ebin/z_logger.beam
** Found 1 name clashes in code paths
```

z_logger has been moved from its own reps/z_logger repo into webzmachine. You can delete the entire `deps/z_logger` directory.

Upgrading to Zotonic 0.8

Module versioning

From 0.8, modules have a schema version concept, which is used to install and update module-specific data (like managed tables, custom categories, default data). Previously this was either done in the module's `init()` or `datamodel()` function. The `datamodel/1` function is no longer called upon module start.

Instead, modules export a `-module_schema()` attribute which contains an integer number, denoting the current module's version. On module initialization, `Module:manage_schema/2` is called which handles installation and upgrade of data. See *Modules* (page 39) for more information and example code.

mod_mailinglist

The mailinglist has changed a bit. You need to manually enable the `mod_logging` module on upgrade. It should be enabled automatically, but please double-check.

Execute the following query to get email sending working:

```
alter table mailinglist_recipient add column is_bounced boolean not null default␣
↪false;
```

Upgrading to Zotonic 0.7

Removed modules

To make Zotonic more lightweight and remove some of the build dependencies, some infrequently used modules have been removed from the core and moved to their own repository, at http://code.google.com/p/zotonic-modules/. These modules are:

- mod_search_solr
- mod_pubsub
- mod_slideshow
- mod_broadcast
- mod_imageclipper
- mod_admin_event
- mod_calendar
- mod_emailer*

All modules, except mod_emailer can still be easily installed with the help of the `zotonic modules install` command. The mod_emailer module (and its esmtp library) has been removed in favor of the native SMTP sending/receiving capabilities.

New SMTP architecture

The mod_emailer module has been removed in favor of a separate mail server process and queueing system. For more information please read the e-mail configuration page in the documentation.

The `emailq` table has become obsolete. You can remove the table from your existing Zotonic database.

Admin password

The admin password is now hardcoded in your site's config file. For sites that are upgrading, you have to add a line to your config file:

```
{admin_password, "letmein"}
```

The value in the config file always reflects the current admin password (as opposed to zotonic < 0.6!) and thus the admin password can only be changed by changing it there.

Admin extra richtext fields

If you have extra richtext (tinymce) fields in the admin, you need to rename the class tinymce of the textarea to the class name tinymce-init.

Upgrading to Zotonic 0.6

No notable upgrade measures need to be taken.

Upgrading to Zotonic 0.5

Some filters disappeared and changed into expression syntax: `|eq`, `|ne`, `|lt`, `|gt`, `|not`, etc.:

`{% if id|eq:2 %}` becomes `{% if id == 2 %}` `{% if id|not %}` becomes `{% if not id %}` et cetera.

The meaning of the query filters *hassubject*, *hasobject*, *hassubjectpredicate* and *hasobjectpredicate* has been reversed:

```
m.search[{query hasobject=id}]
```

becomes:

```
m.search[{query hassubject=id}]
```

and reverse:

```
m.search[{query hasobjectpredicate=id}]
```

becomes

```
m.search[{query hassubjectpredicate=id}] (and reverse)
```

`resource_staticfile's root` directory has changed from the site's template folder to the sites base folder, e.g. from *site/templates/xx* to *site/xx*.

The *m_group'* model no longer exists.

When you first install zotonic and want to logon into /admin, you dont need to give a password, just the username, 'admin'. It will then ask you to set the admin password.

User accounts need to be published otherwise their logon will be denied. Use this query to enable every user in the database:

```
update rsc set is_published=true
where category_id in
        (select distinct(id) from rsc where name='person')
```

If you have an overruled base template, make sure that a {% block content_area %} that spans the full width if your site is in there, because this is used to render the logon dialog for the admin.

Extensions

Zotonic has an extra mechanism for starting additional things that need to be running under the `zotonic_sup` supervisor. These are called *extensions*. Extensions are not tied to any particular Zotonic site, but are regular Erlang processes, in OTP fashion.

On the startup of Zotonic, the global `$ZOTONIC/priv/extensions` folder is scanned once for folders starting with `ext_`. Each of these folders is considered an *extension*.

An extension (for instance, `ext_foo/ext_foo.erl`) can be a regular Erlang module, but is supposed to be something supervisable like a gen_server. It needs to expose at least a function `start_link/0`. On Zotonic startup, this function is called and the resulting pid is added to the Zotonic supervision tree as a *worker* process.

CHAPTER 3

Cookbook

A collection of recipes that show you how to solve reoccurring problems.

Cookbooks

Create a custom action

Todo

This chapter

Create a custom controller

Zotonic comes with a large collection *controllers* (page 11) that cover many use cases, so you'll probably have to resort to custom controllers less often than you may be used to from other web frameworks. Still, the time may come when you need to process HTTP requests in your own way.

You can do so by creating a custom *controller* (page 11). Create a module in your site's `controllers/` directory and prefix it with `controller_`:

Listing 3.1: yoursite/controllers/controller_say_hello.erl

```
-module(controller_say_hello).

-export([
    html/1
]).

%% Include this to have a working controller out of the box
-include_lib("controller_html_helper.hrl").

%% This function renders some HTML when the controller is called
html(Context) ->
    {<<"Hello world and all the people in it!">>, Context}.
```

To be able to handle HTTP requests with this controller, you need to *define a dispatch rule* (page 12) that maps some request URL to this controller:

Listing 3.2: yoursite/dispatch/dispatch

```
[
    {say_hello, ["hello"], controller_say_hello, []}
].
```

Now, if you go to `http://yoursite.com/hello` in your browser, you will see the output of the `controller_say_hello` controller.

Handling POST requests

Now you've seen how to handle GET requests, let's turn to POST requests. First, your controller must show that it can handle POSTs. You do so by adding an `allowed_methods/2` function:

Listing 3.3: yoursite/controllers/controller_say_hello.erl

```
-module(controller_say_hello).

-export([
    %% ...
    allowed_methods/2
]).

%% ...

%% This controller will handle only GETs and POSTs
allowed_methods(ReqData, State) ->
    {['GET', 'POST'], ReqData, State}.
```

A `process_post/2` function will be called with the POST data, so define it:

Listing 3.4: yoursite/controllers/controller_say_hello.erl

```
-module(controller_say_hello).

-export([
    allowed_methods/2,
    html/1,
    process_post/2,
]).

-include_lib("controller_html_helper.hrl").

allowed_methods(ReqData, State) ->
    {['GET', 'POST'], ReqData, State}.

html(Context) ->
    {<<"Hello world and all the people in it!">>, Context}.

process_post(ReqData, Context) ->
    %% Process the POST data
    Context1 = ?WM_REQ(ReqData, Context),
    Context2 = z_context:ensure_qs(Context1),
    Name = list_to_binary(z_context:get_q(name, Context2)),

    {{halt, 200}, wrq:set_resp_body(<<"Thank you posting, ", Name/binary>>,␣
↪ReqData), Context}.
```

Try it out on the command line:

```
$ curl -v -X POST -d 'name=David' http://yoursite.dev/hello

# prints:
Thank you posting, David
```

See also:

- *Controllers* (page 11) in the Developer Guide

Create a custom model

In this chapter we will look at how to implement a model around the The Open Movie Database (OMDB) API.

We will touch useful background information at the same time.

Model modules

Models are Erlang modules that are prefixed with `m_` and stored in your main module's subdirectory *models*. For example, the model to access Zotonic resources (syntax `m.rsc.property`) is written in `models/m_rsc.erl`.

Each model module is required to implement 3 functions (as defined behavior in `gen_model.erl`):

- m_find_value/3
- m_to_list/2
- m_value/2

m_find_value

This function fetches a value from a model. Because there are quite some different variation how to use this function, it is good to understand a bit more about the inner workings of data lookup.

Let's start with the parsing of a template expression:

```
{{ m.rsc[1].is_cat.person }}
```

If you have done some work with Zotonic, you will be familiar with this syntax where we can find out if the resource with id `1` (the Administrator) is of category Person.

This expression contains 4 parts (separated by dots).

At the very start, the template parser resolves `m.rsc` to module `m_rsc`. The parser then calls `m_rsc:m_find_value(Key, Source, Context)` to fetch the value (where Key in our expression has the value of "1").

This is the function specification of `m_find_value`:

```
-spec m_find_value(Key, Source, Context) -> #m{} | undefined | any() when
    Key:: integer() | atom() | string(),
    Source:: #m{},
    Context:: #context{}.
```

It takes a Key and a data Source - a simple record containing 2 entities, model and value:

```
-record(m, {model, value}).
```

`m_find_value` often simply returns a value, but it may also return a (modified) data source record for the next call to `m_find_value`. `m_rsc.erl` resolves our expression in 3 different calls to `m_find_value`, where the data source record ("m" record) is used for pattern matching.

- Step 1: The m record does not contain a value yet. The key (Id) is checked for visibility, and stored in the m record:

```
m_find_value(Id, #m{value=undefined} = M, Context) ->
    ...
    M#m{value=RId};
```

- Step 2: With the m record now containing an Id value, the key `is_cat` is found. Again the key is stored in the m record:

```
m_find_value(is_cat, #m{value=Id} = M, _Context) when is_integer(Id) ->
    M#m{value={is_cat, Id}};
```

- Step 3: The next key to parse is `person`. Since this could have been any category name, a generic `Key` variable is used instead of an atom. The result is calculated in a function call, and returned for further manipulation (e.g. filters) or as string to the page:

```
m_find_value(Key, #m{value={is_cat, Id}}, Context) ->
    is_cat(Id, Key, Context);
```

m_to_list

The second mandatory function transforms a value to a list:

```
-spec m_to_list(Source, Context) -> list() when
    Source:: #m{},
    Context:: #context{}.
```

Not all data models will need to handle lists - in that case the return value is simply the empty list.

Search results are a good example when to apply this function:

```
m_to_list(#m{value=#m_search_result{result=undefined}}, _Context) ->
    [];
m_to_list(#m{value=#m_search_result{result=Result}}, _Context) ->
    Result#search_result.result;
m_to_list(#m{}, _Context) ->
    [].
```

Empty models or undefined results return the empty list; valid results are lifted from its record wrapper and returned as a list.

For example, the `length` filter makes use of this. It calls `erlydtl_runtime:to_list` that calls the model's `m_to_list`:

```
length(Input, Context) ->
    erlang:length(erlydtl_runtime:to_list(Input, Context)).
```

m_value

The final mandatory function specification:

```
-spec m_value(Source, Context) -> undefined | any() when
    Source:: #m{},
    Context:: #context{}.
```

The intended use is to normalize a value to something printable, but you can safely ignore this and return `undefined`.

Example: Setting up m_omdb

All data calls to the OMDB go through the url `http://www.omdbapi.com/?`, with query string appended. We can pass the movie id, title, and pass a type (movie/series/episode). OMDB offers more parameters but we don't need them now.

Template interface

Let's define how will we use the data in templates.

To get all data for a particular ID:

```
m.omdb["tt1135300"]
```

... so that we can get properties like the movie title:

```
{{ m.omdb["tt1135300"].title }}
```

Find an item by title:

```
{{ m.omdb["Alien"].year }}
```

Get all data from a movie:

```
{% for k,v in m.omdb.movie["Alien"] %}{{ k }}: {{ v }}{% endfor %}
```

Get data from a series:

```
{{ m.omdb.series[{query title="Dollhouse"}].plot }}
```

or from an episode:

```
{{ m.omdb.episode[{query title="Dollhouse"}].plot }}
```

Model skeleton

We will write our model in module `models/m_omdb.erl`. Let's first get the mandatory elements out of the way:

```
-module(m_omdb).
-behaviour(gen_model).

-export([
    m_find_value/3,
    m_to_list/2,
    m_value/2
]).

-include_lib("zotonic.hrl").

% ... We will add our m_find_value functions here

% ... Before ending with the final fallback:
m_find_value(_, _, _Context) ->
    undefined.

% This is the default m_to_list if we don't have any list values.
% We will come back to this in a minute
m_to_list(_, _Context) ->
    [].
```

```
% The function that we can ignore
m_value(_, _Context) ->
    undefined.
```

Querying the API

Before diving into the lookup functions, let's see what we want to achieve as result.

1. Using `m_find_value` we will generate a list of query parameters, for example `[{type, "series"}, {title, "Dollhouse"}]`
2. And pass this list to a "fetch data" function
3. That creates a URL from the parameters,
4. loads JSON data from the URL,
5. and transforms the JSON into a property list

The `fetch_data` function:

```
-spec fetch_data(Query) -> list() when
    Query:: list().
fetch_data([]) ->
    [{error, "Params missing"}];
fetch_data(Query) ->
    % Params title or id must be present
    case proplists:is_defined(title, Query) or proplists:is_defined(id, Query) of
        false -> [{error, "Param id or title missing"}];
        true ->
            % Translate query params id, title and type
            % into parameters that OMDB wants
            QueryParts = lists:map(fun(Q) ->
                make_query_string(Q)
            end, Query),
            Url = ?API_URL ++ string:join(QueryParts, "&"),
            % Load JSON data
            case get_page_body(Url) of
                {error, Error} ->
                    [{error, Error}];
                Json ->
                    % Turn JSON into a property list
                    {struct, JsonData} = mochijson2:decode(Json),
                    lists:map(fun(D) ->
                        convert_data_prop(D)
                    end, JsonData)
            end
    end.
```

It is important to know that we will pass a list, and get a list as result (for other template models this may be different).

Lookup functions

To illustrate the simplest `m_find_value` function, we add one to get the API url:

```
-define(API_URL, "http://www.omdbapi.com/?").

% Syntax: m.omdb.api_url
m_find_value(api_url, _, _Context) ->
    ?API_URL;
```

The functions that will deliver our template interface are a bit more involved. From the template expressions we can discern 2 different patterns:

1. Expressions with 1 part:

- `m.omdb["Dollhouse"]`
- `m.omdb[{query title="Dollhouse"}]`

2. Expressions with 2 parts:

- `m.omdb.series["Dollhouse"]`
- `m.omdb.series[{query title="Dollhouse"}]`

When an expression is parsed from left to right, each parsed part needs to be passed on using our m record. For instance with `m.omdb.series["Dollhouse"]` we first tranform "series" to `{type, "series"}`, and then "Dollhouse" to `{title, "Dollhouse"}`, creating the full query `[{type, "series"}, {title, "Dollhouse"}]`.

To parse the type, we add these functions to our module:

```
% Syntax: m.omdb.movie[QueryString]
m_find_value(movie, #m{value=undefined} = M, _Context) ->
    M#m{value=[{type, "movie"}]};

% Syntax: m.omdb.series[QueryString]
m_find_value(series, #m{value=undefined} = M, _Context) ->
    M#m{value=[{type, "series"}]};

% Syntax: m.omdb.episode[QueryString]
m_find_value(episode, #m{value=undefined} = M, _Context) ->
    M#m{value=[{type, "episode"}]};
```

Notice `value=undefined` - this is the case when nothing else has been parsed yet.

The m record now contains a value that will passed to next calls to `m_find_value`, where we deal with the second part of the expression - let's call that the "query" part.

We can either pass:

1. The movie ID: `m.omdb["tt1135300"]`
2. The title: `m.omdb["Alien"]`
3. A search expression: `m.omdb[{query title="Dollhouse"}]`

Luckily, the movie IDs all start with "tt", so we can use pattern matching to distinguish IDs from titles.

For the ID we recognize 2 situations - with or without a previously found value:

```
% Syntax: m.omdb["tt1135300"]
m_find_value("tt" ++ _Number = Id, #m{value=undefined} = M, _Context) ->
    M#m{value=[{id, Id}]};

% Syntax: m.omdb.sometype["tt1135300"]
m_find_value("tt" ++ _Number = Id, #m{value=Query} = M, _Context) when is_
↪list(Query) ->
    M#m{value=[{id, Id}] ++ Query};
```

In both cases we are passing the modified m record. Because we are retrieving a list, we can leave the processing to `m_to_list`. For this we need to update our function:

```
-spec m_to_list(Source, Context) -> list() when
    Source:: #m{},
    Context:: #context{}.
m_to_list(#m{value=undefined} = _M, _Context) ->
    [];
```

```
m_to_list(#m{value=Query} = _M, _Context) ->
    fetch_data(Query).
```

`fetch_data` will return a property list, so we can write this to get all values:

```
{% for k,v in m.omdb["tt1135300"] %}
    {{ k }}: {{ v }}
{% endfor %}
```

Handling the title is similar to the ID. Title must be a string, otherwise it would be a property key (atom):

```
% Syntax: m.omdb["some title"]
m_find_value(Title, #m{value=undefined} = M, _Context) when is_list(Title) ->
    M#m{value=[{title, Title}]};

% Syntax: m.omdb.sometype["some title"]
% If no atom is passed it must be a title (string)
m_find_value(Title, #m{value=Query} = M, _Context) when is_list(Title) ->
    M#m{value=[{title, Title}] ++ Query};
```

To parse the search expression, we can simply use the readymade property list:

```
% Syntax: m.omdb[{query QueryParams}]
% For m.omdb[{query title="Dollhouse"}], Query is: [{title,"Dollhouse"}]
m_find_value({query, Query}, #m{value=undefined} = M, _Context) ->
    M#m{value=Query};

% Syntax: m.omdb.sometype[{query QueryParams}]
% For m.omdb.series[{query title="Dollhouse"}],
% Query is: [{title,"Dollhouse"}] and Q is: [{type,"series"}]
m_find_value({query, Query}, #m{value=Q} = M, _Context) when is_list(Q) ->
    M#m{value=Query ++ Q};
```

Finally, to handle properties like:

```
m.omdb["Alien"].year
```

... we can no longer pass around the m record; we must resolve it to a value and get the property value:

```
% Syntax: m.omdb[QueryString].title or m.omdb.sometype[QueryString].title
% Key is in this case 'title'
m_find_value(Key, #m{value=Query} = _M, _Context) when is_atom(Key) ->
    proplists:get_value(Key, fetch_data(Query));
```

We won't do any validity checking on the parameter here, but for most modules it makes sense to limit the possibilities. See for instance how `m_search:get_result` is done.

Full source code

The source code of the documentation so far can be found in this gist: Zotonic template model for the OMDB movie database - source code to accompany the documentation.

Possible enhancements

For a complete model for this API, I would expect:

- Data caching to speed up identical calls
- Support for all API parameters
- Better error handling (the service might be down or return wrong data)

See also:

- *models section* (page 23) in the Developer Guide
- list of *all models* (page 411).

Custom pivots

Search (page 27) can only sort and filter on *resources* (page 16) that actually have a database column. Zotonic's resources are stored in a serialized form. This allows you to very easily add any property to any resource but you cannot sort or filter on them until you make database columns for these properties.

The way to take this on is using the "custom pivot" feature. A custom pivot table is an extra database table with columns in which the props you define are copied, so you can filter and sort on them.

Say you want to sort on a property of the resource called `requestor`.

Create (and export!) an `init/1` function in your site where you define a custom pivot table:

```
init(Context) ->
    z_pivot_rsc:define_custom_pivot(pivotname, [{requestor, "varchar(80)"}],␣
↪Context),
    ok.
```

The new table will be called `pivot_<pivotname>`. To fill the pivot table with data when a resource gets saved, create a notification listener function `observe_custom_pivot/2`:

```
observe_custom_pivot({custom_pivot, Id}, Context) ->
    Requestor = m_rsc:p(Id, requestor, Context),
    {pivotname, [{requestor, Requestor}]}.
```

This will fill the 'requestor' property for every entry in your database, when the resource is pivoted.

Recompile your site and restart it (so the `init` function is called) and then in the admin under 'System' -> 'Status' choose 'Rebuild search indexes'. This will gradually fill the new pivot table. Enable the logging module and choose "log" in the admin menu to see the pivot progress. Once the table is filled, you can use the pivot table to do sorting and filtering.

To sort on 'requestor', do the following:

```
{% with m.search.paged[{query custompivot='pivotname' cat='foo' sort='requestor'}]␣
↪as result %}
```

Or you can filter on it. The pivot tables are aliassed with a number in order of their occurrence, with the first pivot table aliassed as `pivot1`. This allows you to do filtering on custom fields like this:

```
{% with m.search.paged[{query custompivot="pivotname"
                        filter=["pivot1.fieldname", `=`, "hello"]}]
   as result %}
```

See also:

custompivot (page 32) on how to filter on custom pivot columns.

Create a custom tag

Custom tags, internally called *scomps*, are module-defined tags, which are used when the logic is too complex to be executed in templates.

Custom tags add logic to templates or generate HTML/Javascript constructs that are too difficult for templates. A good example is the scomp-menu scomp which implements the menu, including sub menus -and highlighting of the current menu item.

Implementing a custom tag

A scomp module lives in a module under the `scomps/` folder, and has the following naming convention: `scomp_modulename_scompname`.

So a fictitiuos scomp called `likebutton` in the `mod_facebook` module would defined in the Erlang module called `scomp_facebook_likebutton.erl`.

You can implement scomps by using the *scomp* behaviours. An example scomp that does *absolutely nothing* is implemented as follows:

```
-module(scomp_facebook_likebutton).
-behaviour(gen_scomp).
-export([vary/2, render/3]).
-include("zotonic.hrl").

vary(_Params, _Context) -> nocache.
render(Params, _Vars, Context) ->
    %% FIXME: render like button here
    {ok, Context}.
```

Scomp internals

During the evaluation of scomps we are able to:

- wire actions
- add scripts
- add validators
- add context variables (accessible only by other scomps)

After rendering a template it is inspected to find all scripts, actions and validators. They are placed as #context records in the resulting iolist, the #contexts should be replaced by the 'render' record-attribute. There will be a special 'script' atom to signify the place where all collected scripts will be inserted. When the script atom is not found, then the result of the template is a context, unless there are no collected scripts.

Process state diagram, how all processes work together to render a scomp:

```
Controller        Template                          Scomp server        Scomp
↪Page/Session/User
  |
  |------------>  render(Template,ReqContext)
                      |
                      | ------------------------------- lookup missing var --------
↪-->|
                                                                                 ⌴
↪    |
                      |<------------------------------------ value ---------------
↪---|
                      |
                      |------------------> render(Scomp,ReqContext)
                                                 |
                                                 |--------->⌴
↪render(ReqContext)
                                                 :                |
                                               (if cached)        |
                                                 :                |
                                                 |<--ReqContext--|
                                                 |
                                                 |
                                                 |
```

```
                                                    |
                                                    |
                      |<------------ Output ----------|
                      |
                  Filter scripts
                      |
  |<---- Output ------|
  |
reply user agent
```

The scripts/actions/validators are similar to the ones defined with Nitrogen, though the record structure is redefined to accomodate easy construction by the template compiler.

Writing your own module

Todo

Write this cookbook chapter

Execute tasks asynchronously using the task queue

The Zotonic task queue lets applications perform tasks asynchronously.

Let's say you have some external HTTP API that you want to update whenever a resource in Zotonic is changed. You can so by queuing an task after each resource update. The HTTP request to the external API will be executed asynchronously, so you application and its users do not have have to wait for it.

Add a task to the queue

To add a task to the queue, provide a module and function that should be called when the task is popped from the queue. So to add a task that will call the `external_api_client::update_external_rsc()` function:

```
z_pivot_rsc:insert_task(
    external_api_client,
    update_external_rsc,
    Context
).
```

You can also supply arguments that will be passed to the function. So to have `external_api_client::update_external_rsc(RscId)` called:

```
RscId = 123,
z_pivot_rsc:insert_task(
    external_api_client,
    update_external_rsc,
    undefined,
    [RscId]
    Context
).
```

If you want to queue the task whenever a resource is changed, add this code to an *rsc_update_done* (page 487) observer in your *site module* (page 40):

```
%% yoursite.erl
module(yoursite).

-export([
```

```
    observe_rsc_update_done/2
]).

observe_rsc_update_done(#rsc_update_done{id = RscId}, Context) ->
    z_pivot_rsc:insert_task(
        external_api_client,
        update_external_rsc,
        undefined,
        [RscId]
        Context
    ).
```

Execute queued tasks

Add the function `update_rsc` referenced above to your module:

```
%% external_api_client.erl
-module(external_api_client).

-export([
    update_external_rsc/2
]).

update_external_rsc(RscId, Context) ->
    %% Fetch resource properties
    Json = z_convert:to_json(m_rsc_export:full(Id, Context)),

    %% Execute HTTP query to external API
    {ok, Response} = httpc:request(
        post,
        {
            "https://some-external-api.com",
            [],
            "application/json",
            Json
        },
        [],
        []
    ),

    %% Return anything to signal the task was executed successfully
    ok.
```

Handle failing tasks

The `update_external_rsc` function above assumes that the HTTP request will return successfully. Of course, this is not always the case. To handle failing tasks, you can return a `{delay, NumberOfSeconds}` tuple that will retry the task later:

```
update_external_rsc(RscId, Context) ->

    case httpc:request(
        ...
    ) of
        {ok, Response} ->
            ok;
        {error, Error} ->
            %% Try the task again in one minute
```

```
            {delay, 60}
    end.
```

Prevent duplicate tasks

We decided above that the task should run whenever a resource is changed in Zotonic. However, if a resource is quickly edited multiple times in a row, we only need to send the latest changes once to the external API. In other words, we want to coalesce the tasks into one. You can do so by providing a unique key when queueing the task:

```
UniqueKey = "external-api-" ++ z_convert:to_list(RscId),
z_pivot_rsc:insert_task(
    external_api_client,
    update_external_rsc,
    UniqueKey
    [RscId]
    Context
).
```

Exometer metrics

Zotonic comes with a system for collecting and exporting metrics (such as how much memory is used, how many database requests were made, etc.) called Exometer. This cookbook details how to make use of that system to report on your Zotonic server's activity.

Step 1: get a Time Series Database (TSDB)

At the time of writing, Exometer in Zotonic supports sending metrics to Graphite, OpenTSDB and StatsD, as well as being able to send them to another system using AMQP or polling through SNMP. The installation of such a system is outside the scope of this document, but if you want to experiment without setting up a TSDB for yourself, Hosted Graphite has a 14-day free trial to play with.

Step 2: configure Exometer to report to your TSDB

After deciding on a TSDB, you need to configure Exometer to connect to the TSDB. This is done by configuring a *reporter* in your erlang.config.

The default erlang.config comes with a number of pre-defined exometer metrics:

```
{exometer, [{predefined, [
    {[erlang, memory], {function, erlang, memory, [], value, []}, []},
    {[erlang, system_info], {function, erlang, system_info, ['$dp'], value,␣
↪[process_count]}, []},
    {[erlang, statistics], {function, erlang, statistics, ['$dp'], value, [run_
↪queue]}, []},
    {[erlang, io], {function, erlang, statistics, [io], match, {{'_', input}, {'_',
↪ output}}}, []}
   ]}
]},
```

To configure a reporter, add a block called {reporters, []} and a block inside that for the reporter. For example, to report to a graphite server, the following configuration could be used:

```
{reporters, [
    {exometer_report_graphite, [
        {connect_timeout, 5000},
        {prefix, "zotonic"},
        {host, "graphite.server.com"},
```

```
        {api_key, ""},
        {port, 2003}
    ]}
]}
```

For testing, it is also possible to simply send the output to the console. For this, use the `exometer_report_tty` reporter, like so:

```
{reporters, [
    {exometer_report_tty, []}
]}
```

Other reporters will require different configuration. For more examples, see the Exometer documentation. It is possible to configure multiple reporters at the same time; you can then select what metrics are sent to what reporter with subscriptions.

Step 3: tell Exometer what to report, and how often

Now that Exometer knows *where* to send our metrics, we need to tell it *what* to send, and when to do it. This is done with a *subscription* and, like reporters, can be configured in erlang.config in a block called {subscribers, []}. The general format of a subscription is:

```
{reporter, metric, datapoint, interval}
```

Metrics have a hierarchical name consisting of a list of terms plus a datapoint name. For example, the number of requests made to Zotonic's status page is named:

```
[zotonic, zotonic_status, webzmachine, requests]
```

and the datapoint name is 'value'. So to report this metric to graphite every minute, we would add a subscription as follows:

```
{subscribers, [
    {exometer_report_graphite, [zotonic, zotonic_status, webzmachine, requests],␣
↪value, 60000}
]}
```

Since the metric name includes the site name, it may be preferable to set up generic subscriptions instead, that would apply to all sites in the system. This can be done by using a special syntax in place of the metric name:

```
{select, [{matchspec}]}
```

Where `matchspec` is an ETS Match Specification. The specifics of the Match specification can be quite complex, but we can use simple ones to get going. To select all zotonic metrics of type counter and send them to graphite every minute, you would enter the following:

```
{exometer_report_graphite, {select, [{ {[zotonic | '_'], counter, '_'}, [], ['$_']␣
↪}]}, value, 60000}
```

To send only metrics for a specific site, you can add the site name:

```
{exometer_report_graphite, {select, [{ {[zotonic, sitename | '_'], counter, '_'},␣
↪[], ['$_'] }]}, value, 60000}
```

Using the Zotonic Shell

It is possible to set up exometer from the Zotonic shell. As this run-time configuration is lost upon restart, it is best to only use this for testing or to enact configuration file changes without a restart. A number of uses for the shell are shown below.

To test a match specification. Take the matchspec part of the metric name and feed it to exometer:select(). For example:

```
(zotonic001@server)2> exometer:select([{ {[zotonic, zotonic_status | '_'], counter,
↪ '_'}, [], ['$_'] }]).
[{[zotonic,zotonic_status,db,requests],counter,enabled}, ...
```

To find out what datapoints a metric contains, you can ask for its information:

```
(zotonic001@server)2> exometer:info([zotonic,zotonic_status,webzmachine,requests],␣
↪datapoints).
[value,ms_since_reset]
```

To test a subscription, you can manually add it:

```
(zotonic001@server)2> exometer_report:subscribe(exometer_report_tty, {select, [{
↪{[zotonic, zotonic_status | '_'], counter, '_'}, [], ['$_'] }]}, value, 60000).
ok
```

Complete configuration example

A complete configuration might look like the following:

```
{exometer, [
    {predefined, [
        {[erlang, memory], {function, erlang, memory, [], value, []}, []},
        {[erlang, system_info], {function, erlang, system_info, ['$dp'], value,␣
↪[process_count]}, []},
        {[erlang, statistics], {function, erlang, statistics, ['$dp'], value, [run_
↪queue]}, []},
        {[erlang, io], {function, erlang, statistics, [io], match, {{'_', input}, {
↪'_', output}}}, []}
    ]},
    {reporters, [
        {exometer_report_graphite, [
            {connect_timeout, 5000},
            {prefix, "zotonic"},
            {host, "graphite.server.com"},
            {api_key, ""},
            {port, 2003}
        ]}
    ]},
    {subscribers, [
        {exometer_report_graphite, {select, [{ {[zotonic | '_'], counter, '_'}, [],
↪ ['$_'] }]}, value, 60000},
        {exometer_report_graphite, {select, [{ {[zotonic | '_'], gauge, '_'}, [], [
↪'$_'] }]}, value, 60000},
        {exometer_report_graphite, {select, [{ {[erlang, memory], '_', '_'}, [], ['
↪$_'] }]}, value, 60000},
    ]}
]},
```

Frontend cookbook

These cookbook entries contain valuable nuggets of information regarding the frontend development of a site.

Implementing a simple contact form

This tutorial teaches you to create a form, validate it, submit it over Ajax and e-mail the results back to you.

Why

Making a simple contact form might seem difficult, but with the smart application of different Zotonic techniques you'll see that it's actually very easy.

1. Create the contact page URL dispatcher and template
2. Create the contact form
3. Create the contact-form handler Erlang file.

Assumptions

Readers are assumed to be comfortable both in developing Zotonic templates and in writing Erlang modules.

How

Create the contact page URL dispatcher and template

The URL dispatcher is placed in `user/sites/yoursite/dispatch/dispatch`. Add this line:

```
{contact_url, ["contact"], controller_template, [ {template, "contact.tpl"} ]},
```

This says that the page at "/contact" will use the "contact.tpl" template. Let's create this template, at `user/sites/yoursite/templates/contact.tpl`:

```
{% extends "base.tpl" %}

{% block content %}
<h1>Contact page</h1>
{% endblock %}
```

Now we have this, let's try to see if it loads. Flush the Zotonic cache (to refresh the URL dispatchers) by going to "modules" -> "rescan modules" in the admin. Now, point your browser to http://yoursite:8000/contact. This should show the contact page with the template you just made.

Note: Your actual site location might be different, see the *User sites directory*.

Create the contact form

Now you should write the acual contact form. You should decide what fields you want in the form, so for now, just put a name, e-mail and comment field:

```
{% wire id="contact-form" type="submit" postback={contact} delegate="my_contactform
↪" %}
<form id="contact-form" method="post" action="postback">

  <label for="name">Name</label>
  <input type="text" name="name" id="name" />

  <label for="email">E-mail</label>
  <input type="text" name="mail" id="mail" />
  {% validate id="mail" type={presence} type={email} %}

  <label for="message">Message</label>
  <textarea name="message" id="message" cols="60" rows="8"></textarea>
  {% validate id="message" type={presence} %}
```

```
  <input type="submit" value="Send" />
</form>
```

This form has 3 fields, of which the message and the e-mail are required, and the e-mail input has to contain a valid e-mail address. The name field is optional.

Create the contact-form handler Erlang file

As you see in the scomp-wire statement in the contact form, the *delegate* argument is set to `my_contactform`, which is the name of an erlang module which we still have to create. When the form submits, this module's event/2 function gets called. Create a file `user/sites/default/support/my_contactform.erl` with the following contents:

```
-module(my_contactform).
-export([event/2]).

-include_lib("zotonic.hrl").

event({submit, {contact, []}, _TriggerId, _TargetId}, Context) ->
  ?DEBUG(z_context:get_q_all(Context)),
  Context.
```

This is the most simple version of a Zotonic form-handling function: it just dumps all form input it gets to the Zotonic console (using the ?DEBUG macro). To compile this Erlang module, enter the following on the zotonic console:

```
z:m().
```

You'll see a notice that the file is recompiled, which should end with a `ok` message to indicate the success. This compiling is actually very important: Whenever you change the .erl file, you'll need to recompile it using this command.

E-mail the contents of the contact form to somebody

Using Zotonic's email module, you can very easily send somebody an e-mail. Let's create a simple template to send the contents of the form to the site administrator.

Create the file `user/sites/default/templates/_email_contact.tpl`:

```
<html>
  <head>
    <title>Contact form</title>
  </head>
  <body>
    <p>Hello, the contact form of the site has been submitted.</p>
    <p>Name: {{ name|escape }}</p>
    <p>E-mail: {{ mail|escape }}</p>
    <p>The contents of the message was this:</p>
    <pre>{{ message|escape }}</pre>
    <p>Regards, your website.</p>
  </body>
</html>
```

This template will function as the message body that will be sent. Note: this template gets scanned for the <title> tag, which will double as the e-mail's subject, so be sure to include it!

Now we have to change our `event/2` function to render this template and e-mail it using mod_emailer. Change the event function to the following:

```
event({submit, {contact, []}, _TriggerId, _TargetId}, Context) ->
  Vars = [{mail, z_context:get_q("mail", Context)},
          {name, z_context:get_q("name", Context)},
          {message, z_context:get_q("message", Context)}],
  z_email:send_render(z_email:get_admin_email(Context), "_email_contact.tpl", Vars,
↪ Context),
  z_render:update("contact-form", "<p>The form has been submitted! Thank you, we
↪'ll get in touch soon.</p>", Context).
```

This loads the relevant values from the form, puts them in the Vars variable, and then calls the z_email module to mail the given template to the e-mail address of the site admin (which is defined in your site's config file). For more information on sending mails from Zotonic, please see the mod_emailer documentation.

Finally, this contact-form handler replaces the contact form with a <p> tag with a success message, using the `z_render:update` function.

How to add a custom Content Block

Zotonic comes with a number of standard content blocks: Header, Text and Embed page. Additional content blocks are provided by modules, for example mod_survey uses content blocks extensively for composing surveys. Content blocks allow a content manager to add sophisticated sections of content, perhaps with configuration options. They could be used for design reasons e.g. for multi-column layouts, image carousels or floating elements. This cookbook item describes how to add a simple custom content block.

In order to add a custom content block, we need to register it, by editing an erlang file and adding two templates. For this tutorial we'll add a content block which inserts the code for a music player from jamendo.com, a Creative Commons music hosting site.

To register the content block we add an `observe_admin_edit_blocks/3` function to the site's erl file e.g. `zotonic/user/sites/mysite/mysite.erl`:

```
%%====================================================================
%% support functions go here
%%====================================================================


-export([
    observe_admin_edit_blocks/3
]).


observe_admin_edit_blocks(#admin_edit_blocks{}, Menu, Context) ->
    [
         {100, ?__("Media", Context), [
             {music_player, ?__("Music Player", Context)}
         ]}
     | Menu
    ].
```

The interesting parts are `"Media"`, `"Music Player"` and `music_player`. `"Media"` will appear as a section heading in the `[+add block]` drop down menu, below the `Standard` section. `"Music Player"` will appear below this, and is what the content editor clicks on to instert the block. The `music_player` atom is what Zotonic uses to figure out which templates to use, they will be created in `templates/blocks` and will be called `_admin_edit_block_li_music_player.tpl` and `_block_view_music_player.tpl`.

As the name suggests, `_admin_edit_block_li_music_player.tpl` is the template used on the edit form:

```
{% extends "admin_edit_widget_i18n.tpl" %}

{% block widget_title %}{_ Block _}{% endblock %}
```

```
{% block widget_show_minimized %}false{% endblock %}
{% block widget_id %}edit-block-{{ #block }}{% endblock %}
{% block widget_header %}{% endblock %}

{% block widget_content %}
    <div class="control-group">
            <label class="control-label">URL of the music widget</label>
            <div class="controls">
                    <input type="text" id="block-{{name}}-url"
                            name="block-{{name}}-url"
                            value="{{ blk.url }}"
                            class="input-block-level"
                            placeholder="{_  Enter the url of the music widget _}"
                    />
            </div>
    </div>
{% endblock %}
```

For the purpose of demonstration, we extend the `admin_edit_widget_i18n.tpl` template and use some template blocks.

Each time this block is added to the page, the form above will be displayed. Here, we want to allow users to easily embed a music widget from Jamedo, an online music community. The way do this is by adding extra input fields to the page through the block templates.

Please note the values of the attributes of the input fields in the block template. For example, `name="block-{{name}}-url"` will expand to `name="block-mus56h-url"`. The `name` variable is a randomly generated string that is unique within every block template. The last portion of the value of the attribute is the attribute to be added to the resource. In this case, the attribute to be added will be `url`.

Also note the value of the input's value attribute. This ensures that if the attribute has already been saved against the resource being edited, then the input will be filled with the saved value. Without this, the data entered will not be saved.

This is all you need to be able to add a block to the edit form. If you update your site and restart, you should now be able to select the new block. It just doesn't display anything yet so let's add `_block_view_music_player.tpl`:

```
<iframe id="widget" scrolling="no" frameborder="0" width="400"
        height="284" style="width: 400px; height: 284px;"
        src="{{ blk.url }}"></iframe>
```

In the block's frontend view template, a special variable called `blk` is available. This `blk` variable holds the attributes or properties of the block. Since we added only one attribute, `url`, to the block's admin template, the `blk` variable will hold only two properties: `name`, and our custom attribute, `url`.

So if the user supplied `http://widgets.jamendo.com/v3/album/40728?autoplay=0&layout=standard&width=400`, as the url to the music widget, the block's frontend view template will expand to:

```
<iframe id="widget" scrolling="no" frameborder="0" width="400"
    height="284" style="width: 400px; height: 284px;"
    src="http://widgets.jamendo.com/v3/album/40728?autoplay=0&layout=standard&
↪width=400">
</iframe>
```

We can extend this custom block to allow the user to specify the widget's height, width, and frameborder.

All you have to do is add new input fields in the block's admin template.

Updating form field from a dialog

Ever wanted to update a form field from a dialog, possibly giving the user some list to choose from? Here's how to do it.

Why

Some interactions benefit from presenting a dialog for input. This guide explains how to take input in a dialog and have it apply to a form input value.

Assumptions

Readers are expected to understand DOM IDs for uniquely identfying form inputs and also understand advanced usage of Zotonic Template tags.

How

To be able to update a form field, you need to pass the id of the element you want to update to the dialog. It may look something like:

```
{% button text="Update field dialog"
        action={dialog_open
                title="Update form field value"
                template="my_dialog.tpl"
                target_id="input_element_id"}
%}
```

Then create your dialog template, my_dialog.tpl in this example, and wire a set_value action to update the input form element:

```
<a id="my_anchor" href="javascript:void(0)" href="javascript:void(0)">Click here␣
↪to update form value</a>
{% wire id="my_anchor"
        type="click"
        action={set_value target=target_id value="My new value"}
        action={dialog_close}
%}
```

If you include a template like the one above into your dialog template many times (i.e. from a for loop), then having fixed ids are no good. To prefix the id with a unique value (per invocation of the template) prefix the id with a #-sign. so the a-tag becomes `<a id={{#my_anchor}}...` and the wire becomes `wire id=#my_anchor...` which will expand to something like `"ubifgt-my_anchor"`.

See also:

Auto-generated identifiers (page 26)

Enabling Growl Notifications

Using growl outside admin requires some magic to make it work.

Why

Growls provide an unobtrusive way of notifying users of background events or the completion of tasks. This guide provides step-by-step instructions on how to enable it for your site.

Assumptions

Readers are expected to be familiar with template editing.

How

Although the magic is quite simple. The missing pieces are a couple of client side scripts, `/lib/js/modules/z.notice.js` and `/lib/css/z.growl.css` from *mod_base* (page 287).

In your `base.tpl` template, include these files using the *lib* (page 453) tag:

```
{% lib "js/modules/z.notice.js" %}
```

And the CSS:

```
{% lib "css/z.growl.css" %}
```

Now you should be able to use growl actions in your templates, example:

{% button action={growl text="hello world"} %}

See also:

action-growl, *lib* (page 453)

Add Chat to Your Zotonic Site

Thanks to Michael Connor's `zchat`, it's easy to add chat system on your Zotonic site.

Why

It is often very useful for a site to have a live support line built-in. Another case for chat is on a social site where you'd like your members to be able to communicate in real time.

Assumptions

Readers are assumed to have a working Zotonic system running, and Mercurial (the `hg` command) installed.

How

Install the `mod_chat` module using `zotonic modules install`:

```
$ zotonic modules install mod_chat
```

This should download `mod_chat` and install it in the *priv/modules/* directory. Restart and rebuild zotonic to see the effect of these changes:

```
$ cd /home/zotonic/zotonic
$ zotonic stop
$ make
$ zotonic start
```

Now, log into your site admin page, select *System* and then *Modules*. Scan down to find mod_chat. Activate it.

Now you can either include chat into an existing page on your site or create a new page.

To include chat in an existing page, find the template that formats the page in the templates directory and somewhere within the content block enter:

```
{% include "_chat_box.tpl" %}
```

To create a new page:

Create a page called "Chat" with category Text. Give it the Unique name `page_chat`, click on the Published check box, and save.

Add the following dispatch rule to your dispatch list:

```
{chat, ["chat"], controller_page, [ {template, "chat.tpl"}, {id, page_chat} ]},
```

If this is the last rule in your list, omit the trailing comma.

Go to *System* and *status* in your admin menu and click on *rescan modules*.

Now you should be able to go to `<yoururl>/chat` and see your chat window. You'll see *Anonymous <long number>* in the right-most pane. That's you. If your chat buddy enters the chat page, a second Anonymous <another big number> will appear. When you or your buddy leave the chat page, the respective Anonymous tag will disappear. Note that chat messages are not stored or logged. That may come with a future version of zchat.

Modifying an existing module

Safely overriding core modules to make them do what you need or want.

Why

Sometimes a module presents things on screen in a way you don't like. It might be that a certain piece is missing or the order of the elements breaks something you are trying to do. This guide presents some techniques for working around this.

Assumptions

Readers are assumed to be comfortable editing templates and identifying the source of content by searching the Zotonic codebase.

How

If you want to just change the look and feel, you should be able to get most of the way with CSS: create a CSS file in your site, and override the styles you don't like.

Failing that, you should try overriding templates: Copy the module's templates into your own site/module, keeping the file name and relative directory intact.

Make sure that the *priority* of your site is higher than the original module! To view this, check the *prio* field in the module overview in the admin, this should be a lower number. e.g. 1 has higher prio than 500.

Changing the Logic of a Module

In extreme cases, you can copy the module into the modules directory of your site and rename it. Make sure to change the module attributes within the Erlang modules along with renaming the files themselves.

A case for this would be an outdated core module that no longer works with an API or protocol due to external changes. For example, if Twitter dropped their current streaming interface (unlikely) then you might need to copy and modify mod_twitter in order to bridge the gap until you submit a patch and its gets accepted.

Page-specific dynamic backgrounds

Use edges (*page connections*) to associate backgrounds with pages.

Contributed by: Dmitrii Dimandt

Why

This is a small use case with a lot of words pouring into it.

I'm doing a very small project for a couple of friends. It currently resides on http://rusukr.dmitriid.com/ and will soon move to a proper domain.

What the project needed was different backgrounds for different pages. Compare, for instance, http://rusukr.dmitriid.com/products and http://rusukr.dmitriid.com/investments

Assumptions

Readers are expected to be familiar with where templates reside and how things like conditional tags work. To benefit most you should also be comfortable getting around and editing items in Zotonic's CMS admin interface.

How

Now, the most obvious way to implement these backgrounds would be to attach a media item to a page and something along the lines of:

```
{% if id.medium %}
  {% media id.medium %}
{% endif %}
```

Actually, if you create your site using the blog skeleton, you will see similar code in *_body_media.tpl*. Obviously, such an approach has a severe downside. What if you actually want to attach other media to the page? What if you want to display images and video in your text, not just a background?

This is where custom predicates come into play. Don't fret! it's much easier than you think

Predicate is nothing but a fancy way to say "relationship". Predicates define relationships between various places (objects, entities) within Zotonic. If you are into graphs and graph theory, a predicate is a directed labeled edge from one node to another (see Marko Rodriguez's amazing presentations on (among oher things) graph theory and graph algebra here: http://www.slideshare.net/slidarko/presentations).

What a predicate does, it defines how objects relate to other objects. Since predicates are (usually) one-way only, you helps to think about them as follows:

if I have a parent, how do I discern between its children? Or how do I define its children? if I have an object, how can I have different collections of objects related to it? So, if you have an article A, then you can have:

- a number of tags describing the article
- several authors who collaborated on it
- a few images to go with the article

Let's rephrase that:

- an article has tags
- an article has authors
- an article has images

Once you can make such a "has" connection, you can use a predicate:

```
article -----tag predicate---> tag1, tag2, tag3, etc.
article -----author predicate---> author1, author2, author3, etc.
article -----image predicate---> image1, image2, image3, etc.
```

Easy, isn't it?

The greatest thing about predicates is that you can define your own. Here's how yu do it.

- Go to the "Predicates" section (under *Structure*) and click "Make a new predicate".
- Name it "Background", since its used to signify a dynamic background on a page.
- Click "Make predicate".

On the following screen we have to choose the direction of our predicate. "From" is the entity that can have relations to other objects, and "To" is the entity that can be had . So, our Text has an Image as a background. This means that "From" is Text and "To" is Image. Click the corresponding checkboxes and then click save.

you're done!

Now if you create a new text item (an article or news) you will see that in the Page Connections section there's now a new option, "Background". Let's try and make a new background:

- Go to Pages
- Click "Make a new media item"
- Name this item, for example, "test"
- Select an image to upload from your computer
- Click "Upload file"
- The page that opens is irreleveant to us right now. Just go back to "Pages".
- Click "Make a new page"
- Name this page, for example, "test page"
- Click on "Background" in "Page connections section"
- Type in "test"
- Select "test" media item
- Click "Save and view"

You should now see your test page... with no background on it. This is ok, since we haven't told Zotonic how we want to display our background. To do this we have to edit a template. Let's try and display the background image first. First, we can make a checklist to see how we should proceed:

- We need to make sure the background exists
- We need to retrieve the resource associated with the background
- We need to retrieve path to the image stored on the server
- We need to make sure that this path is actually accessible from a web browser

The last step is especially important since Zotonic stores uploaded files in a directory that is not directly accessible from the web. However, for images we can use the *image_url* (page 451) tag to circumvent that.

So, the code to go with our checklist is as follows:

```
<div{% if id.background %}{# we check to see if background exists #}
    style="background: url( {% image_url id.background.medium.filename %}{# output␣
↪web accessible URL to the image #} ) no-repeat"
    {% endif %}>
   
</div>
```

Now that we know how to retrieve the background image we can use it to our advantage. Our dynamic background will now look something like this:

```
<div{% if id.background %} style="background: url({% image_url id.background.
↪medium.filename %})"{% endif %}>
   
</div>
```

Retrieving the category of a page

Getting the category from a URL is somewhat involved, but not impossible. This is an example of what you can do with filters.

Why

It is often useful to use a page's category to present it on the page itself or to look up related content. This guide provides step-by-step instructions for getting a page's category in a template.

I have a page with url of the form `/my_category`, `/my_category/:id`, or `/my_category/:id/:slug`. How can I retrieve the category from the url?

Assumptions

Readers are assumed to be comfortable with template development.

How

Since category is a direct property of m_rsc we can directly access it:

```
{% with id.category as my_cat %}
  ...
{% endwith %}
```

Share variable binding across blocks

How to avoid having to call the same query inside several blocks of the same page

Why

In some situations, you may have to use the same query result inside several blocks of the same template. For instance, you may need to use it in the body of a page and in its JavaScript block. Intuitively, you would program your template as shown in these scripts:

```
{# base.tpl #}
<html>
<head>...</head>
<body>
{% block html_body %}
    <!-- This is the default body -->
{% endblock %}
</body>

<script>
{% block js %}
```

```
    // This is the default js
{% endblock %}
</script>

</html>
```

And a page:

```
{# mypage_1.tpl #}
{% extends "base_1.tpl" %}

{# THIS won't WORK BECAUSE %with% AND %block% TAGS CANNOT BE NESTED THIS WAY #}
{% with m.mymodule.myquery as myresult %}

{% block html_body %}
    Here is your result: {{ myresult|escape }}
{% endblock %}

{% block js %}
    alert('Do something with your result: {{ myresult|escapejs }}');
{% endblock %}

{% endwith %}
```

Unfortunately, this doesn't work, because Zotonic doesn't allow you to place a `{% with %}` tag around all `{% block %}` tags of *mypage_1.tpl*.

One solution consists in duplicating the `{% with %}` tag and moving it inside each block:

```
{% extends "base_1.tpl" %}

{% block html_body %}
    {% with m.mymodule.myquery as myresult %}
        Here is your result: {{ myresult|escape }}
    {% endwith %}
{% endblock %}

{% block js %}
    {% with m.mymodule.myquery as myresult %} {# AGAIN THE SAME QUERY #}
    alert('Do something with your result: {{ myresult|escapejs }}');
    {% endwith %}
{% endblock %}
```

However, this has a severe drawback: the same assignment will be done twice, and if it comes from an "expansive" database query, this query will be performed twice (or the query result will have to be cached, which increases the complexity of your system significantly).

Assumptions

Readers are assumed to be comfortable with the template syntax and with template inheritance, specifically the `{% extends %}` and `{% block %}` tags.

How

The solution is to move both blocks to a new template, `_html_body_and_js.tpl`:

```
{# _html_body_and_js.tpl #}

<body>
```

```
{% block html_body %}
    <!-- This is the default body -->
{% endblock %}
</body>

<script>
{% block js %}
    // This is the default js
{% endblock %}
</script>
```

Our base template includes the new template so its contents will be drawn on the page:

```
{# base.tpl #}

<html>

<head>...</head>

{% block html_body_and_js %}
    {% include "_html_body_and_js.tpl" %}
{% endblock %}

</html>
```

Now we make sure that `_html_body_and_js.tpl` gets data. In a base subtemplate `smart_base.tpl` we override the block with the query:

```
{# smart_base.tpl #}

{% extends "base.tpl" %}

{% block html_body_and_js %}
    {% with m.mymodule.myquery as myresult %}
        {% include "_html_body_and_js.tpl" %}
    {% endwith %}
{% endblock %}
```

The page that needs the data extends `smart_base.tpl`. Variable `myresult` is now accessible and can be used in both blocks:

```
{# mypage.tpl #}
{% extends "smart_base.tpl" %}

{% block html_body %}
    Here is your result: {{ myresult|escape }}
{% endblock %}

{% block js %}
    alert('Do something with your result: {{ myresult|escapejs }}');
{% endblock %}
```

Customizing the sign up and sign in form

You want to change parts of the form, or change its appearance.

Sign up form

The sign up form is called with dispatch rule `signup`, implemented by *mod_signup* (page 311). This module must be enabled to view the form.

The form is built from quite a number of sub templates. While this creates some level of complexity (to find out what goes where), this also provides the flexibility to change the resulting output without breaking the functional code.

For details, see "Template structure Sign up form" below.

Overriding templates

Some sub templates are more critical than others.

While it is technically possible to override the sub templates, this will likely end up with non-working pages. The Page Controller expects to receive certain values from the submitted form.

These form fields are expected to have a value:

```
email
name_first
name_surname
username
password1
signup_tos_agree
```

If you want to hide a part of the form that contains a critical value, for instance the terms and conditions checkbox, you cannot just create an empty file `_signup_form_fields_tos.tpl` in your project, because the form value `signup_tos_agree` will no longer be passed.

Instead you need to pass the value through the form using a hidden input field:

```
<input type="hidden" name="signup_tos_agree" value="1" />
```

Non-critical templates can be overridden safely. For instance `_signup_title.tpl` will only change the title above the form, so this can be done without affecting the functionality. You could decide to add a brand logo to the title, or remove the title altogether.

Changing the configuration

The template file `_signup_config.tpl` is made to quickly define what parts of the sign up page should be shown. Copy the file from `mod_signup` to your project and change the values.

For convenience, critical variables are not exposed in the config file.

By default, two form parts are set to hidden. Make them visible by changing the value from 0 to 1:

```
show_signup_name_prefix          // surname prefix ("van" Gogh)
show_signup_password2            // repeat password
```

Two other configuration values deal with appearance: `style_boxed` and `style_width` - see below.

Changing appearance

Form fields are displayed using the Bootstrap 3 HTML structure. If you need a radical different layout that cannot be managed with CSS alone, you will need to override the form field templates.

The width of the form can be set with configuration value `style_width`. This should be a CSS value like "300px". If no value is set, the form will expand to maximum width - in which case the width needs to be set by CSS.

A form background can be set with configuration value `style_boxed`.

CSS styles are defined in `css/logon.css` (mod_authentication).

Password length

The minimum password length is 6 characters, and can be changed in config value `mod_authentication.password_min_length`.

Sign up form in a modal dialog

Instead of directing the user to the sign up page, the form can be shown in a modal dialog:

```
<a id="{{ #signup }}" href="#">{_ Sign up _}</a>
{% wire
    id=#signup
    action={
        dialog_open
        title="Sign up"
        template="signup.tpl"
    }
%}
```

However, this does not work well if the user wants to switch to sign in: clicking the link will load the sign in page, removing the dialog.

A better approach is to use the sign in templates and pass `logon_state`:

```
<a id="{{ #signup }}" href="#">{_ Sign up _}</a>
{% wire
    id=#signup
    action={
        dialog_open
        title="Sign up"
        template="logon_modal.tpl"
        logon_state="signup"
    }
%}
```

If the user now clicks on the link to sign in, the new form is shown in the same dialog.

Sign in form

For a general understanding of the templates and configuration, first read the section on Sign Up form above.

The sign in form is called with dispatch rule `logon`, implemented by *mod_authentication* (page 286). This module will normally be enabled.

Sign In covers a number of associated functions:

- Sign in (both public and admin)
- Request password reset
- Feedback after reset
- Create new password
- Feedback when account needs verification

This makes the template structure more complex than the sign up form.

Overriding templates

The templates are quite minimal and will probably not need structural changes. Most likely candidates for changing are titles and perhaps removing/adding extra links.

For details see "Template structure Sign Up form" below.

Changing the configuration

The two configuration values in template file `_logon_config.tpl` deal with appearance: `style_boxed` and `style_width` (see below). Copy the file from `mod_authentication` to your project and change the values.

Changing appearance

Form fields are displayed using the Bootstrap 3 HTML structure. To change the layout, look at the form field templates:

- _logon_login_form_fields.tpl
- _logon_reminder_form_fields
- _logon_reset_form_fields

The width of the form can be set with configuration value `style_width`. This should be a CSS value like "300px". If no value is set, the form will expand to maximum width - in which case the width needs to be set by CSS

A form background can be set with configuration value `style_boxed`.

CSS styles are defined in `css/logon.css` (mod_authentication).

Reference: Template structure Sign up form

Template tree:

```
signup.tpl                                  // sign up page
`-- _signup_config.tpl                      // template and field configuration
    `-- _signup.tpl                         // if signed in, redirects to user␣
↪page
        `-- _signup_box.tpl                 // form box components
            `-- _signup_stage.tpl           // feedback message
```

The central form template `_signup_box.tpl` is further populated by sub templates:

```
_signup_box.tpl
|-- _signup_title.tpl                       // header "Sign up"
|-- _signup_extra.tpl                       // sign up with other auth modules␣
↪(if activated)
|-- _signup_form_form.tpl                   // HTML form
|   `-- _signup_form_fields.tpl             // 3 form parts plus submit button
|       |-- _signup_form_fields_email.tpl   // name and email
|       |-- _signup_form_fields_username.tpl // username and password
|       `-- _signup_form_fields_tos.tpl     // terms of service
|-- _signup_support.tpl                     // left empty
|-- _signup_outside.tpl                     // link (back) to sign in
`-- _logon_link.tpl                         // shown below sign in form
```

Reference: Template structure Sign Up form

Template tree:

```
logon.tpl                                       // sign in page
|-- _logon_config.tpl                           // template and field configuration
|   `-- _logon.tpl or _logon_modal.tpl          // logon module
|       `-- _logon_box.tpl                      // form box components
|           `-- _logon_stage.tpl                // feedback messages
|               `-- _logon_expired_form.tpl     // when pw is expired
`-- logoff.tpl                                  // log off page, redirects to q.p or␣
↪homepage
```

Two mechanisms handle the state to determine which sub templates should be read:

- For display on the page: the dispatch rule
- For display inside a modal: the `logon_state` value

Depending on the the state value, `_logon_box.tpl` is populated by different sub templates:

```
when logon_state is::
|== logon_reminder:                            // request a pw reset
|   `-- _logon_box.tpl
|       |-- _logon_reminder_title.tpl
|       |-- _logon_reminder_form.tpl
|       |   `-- _logon_reminder_form_fields.tpl or _logon_reminder_admin_form_
↪fields.tpl
|       `-- _logon_reminder_support.tpl         // backlink to logon form
|== logon_reset:                               // reset pw
|   `-- _logon_box.tpl
|       |-- _logon_reset_title.tpl
|       |-- _logon_reset_form.tpl
|       |   `-- _logon_reset_form_fields.tpl
|       `-- _logon_reset_support.tpl            // backlink to logon form
|== admin_logon:
|   `-- _logon_box.tpl
|       |-- _logon_error.tpl
|       |-- _logon_login_form.tpl
|       |   `-- logon_login_admin_form_fields.tpl
|       |-- _logon_login_support.tpl            // link forgot password
|== signup (logon_state/modal only)
|   `-- _signup_config.tpl (see Sign Up form)
|       |-- _signup_support.tpl                 // backlink to logon form
`== else:
    `-- _logon_box.tpl
        |== awaiting verification:
        |   `-- _logon_stage.tpl                // alternative content for logon box
        `== else:
            |-- _logon_login_title.tpl          // title "Sign in to ..."
            |-- _logon_login_extra.tpl          // all-include by other modules
            |-- _logon_error.tpl
            |-- _logon_login_form.tpl
            |   `-- _logon_login_form_fields.tpl
            |-- _logon_login_support.tpl        // link forgot password
            |-- _logon_login_outside.tpl        // all-include _logon_link.tpl
               `-- _logon_link.tpl              // all-include by other modules
```

Here the sub templates ensure a consistent markup inside the box when going from state to state.

Managing redirection after login and signup

Configure mod_signup to redirect to something other than a member's home page.

Why

The default behavior of Zotonic is to redirect the user to his or her own page after logon (/page). If you want to change this, you need to modify the `sitename.erl` file, where *sitename* is the name of your site. Remember that this file is located in `user/sites/sitename/`.

Note: Your actual site location might be different, see the *User sites directory*.

Assumptions

Readers are assumed to be comfortable with Erlang syntax and use of the command shell.

How

Open `user/sites/sitename/sitename.erl` with your favorite editor:

```
$ cd /path/to/zotonic
$ vim user/sites/sitename/sitename.erl
```

Example 1

Redirection to the /welcome page after signup and to the /welcome-back page after logon

Add the following code:

```
-export([observe_signup_confirm_redirect/2, observe_logon_ready_page/2]).

%% @doc Set the page that must be opened after signup
observe_signup_confirm_redirect({signup_confirm_redirect, _UserID}, _Context) ->
    "/welcome".

%% @doc Set the page that must be open after login
observe_logon_ready_page({logon_ready_page, _}, _Context) ->
    "/welcome-back".
```

Example 2

Conditional redirection, where the site administrator is redirected to the /admin page, and all other users are redirected to the /welcome page:

```
-export([observe_logon_ready_page/2]).

%% @doc Set the page that must be opened after logon
observe_logon_ready_page({logon_ready_page, _}, Context) ->
    case z_acl:user(Context) of
        ?ACL_ADMIN_USER_ID  -> "/admin";
      _                     -> "/welcome"
  end.
```

Example 3

Conditional redirection with one option being the default behaviour:

```
-export([observe_logon_ready_page/2]).

%% @doc Set the page that must be opened after logon
observe_logon_ready_page({logon_ready_page, _}, Context) ->
    case z_acl:user(Context) of
        ?ACL_ADMIN_USER_ID  -> "/admin";
        _                   -> undefined     % All users except the administrator␣
↪are redirected to their own pages
    end.
```

Example 4

Conditional redirection, where the site administrator is redirected to the /admin page and all other users are redirected to the /welcome page, unless another URL is specified:

```
-export([observe_logon_ready_page/2]).

%% @doc Set the page that must be opened after logon
observe_logon_ready_page({logon_ready_page, Url}, Context) ->
    case {z_acl:user(Context), Url} of
        {?ACL_ADMIN_USER_ID, _} ->
            "/admin";
        {_, []} ->
            "/welcome";
        _ ->
            Url
    end.
```

Save your changes and quit.

Recompile zotonic:

```
$ cd /path/to/zotonic
$ make
```

Now when logging in, you should see this redirect behaviour in action.

Displaying a site map

For the benefit of search engines and fans of tables of contents you can easily provide a site map.

Note: This article discusses HTML site maps. For the XML kind which Google uses, see *mod_seo_sitemap* (page 310).

Why

Some users prefer to navigate a site based on its underlying structure. Search engines also use this information to determine how pages relate to each other within your site. For these cases it is useful to have a site map.

Assumptions

Readers are expected to know where templates go in a site and how they basically work. To benefit the most you will also need experience in CSS to style the end-result effectively.

How

This is straight-forward except for the CSS (which isn't provided here), but that isn't entirely avoidable since it depends completely on how you want to display things. On the other hand this does give you a workable nested unordered list with links to the pages which certainly serves the basic functionality out of the box.

Create site_map.tpl containing:

```
{% extends "page.tpl" %}
{% block below_body %}
    {% menu %}
{% endblock %}
```

Create a dispatch rule to take /site-map to your Site Map page:

```
{site_map, ["site-map"], resource_page, [{template, "site_map.tpl"}, {id, page_
↪site_map}]}
```

Create a Page with the Admin interface and set its Unique Name to `page_site_map`. Whatever you put in the body will show above the site map.

I haven't discussed CSS, but the HTML generated by the scomp-menu tag is pretty CSS friendly.

See also:

scomp-menu

Site-specific signup actions

Performing additional, project-specific actions when a user signs up

Why

When a user signs up, Zotonic verifies the validity of some information, such as the e-mail address and the password strength. If these verifications succeed, it stores the user data in the Zotonic database. If you have your own database with project-specific user data, you may want to insert additional information into this database on sign up. To do this, you need to modify the `sitename.erl` file, where *sitename* is the name of your site. Remember that this file is located in `user/sites/sitename/`.

> **Note:** Your actual site location might be different, see the *User sites directory*.

Assumptions

Readers are assumed to be comfortable with Erlang syntax and use of the command shell.

How

Open `user/sites/sitename/sitename.erl` with your favorite editor.

Example 1: a gaming website with several point accounts for each user. Add the following code:

```
-export([observe_signup_done/2]).

%% @doc Perform additional actions when a user signs up
observe_signup_done({signup_done, UserId, _IsVerified, _Props, _SignupProps},
        Context) ->
```

```
    % Create the necessary accounts for the user:
    create_user_accounts(UserId, [chess, go, checkers]).

% Implementation not shown here
create_user_accounts(_UserId, _Context) ->
    todo.
```

Save your changes and quit

Recompile zotonic:

```
$ cd /path/to/zotonic
$ make
```

Now you'll see that when a signup is done successsfully, your `create_user_accounts/2` function will be called.

Dynamic select options using a wired template

Why

Suppose you want to wire a change event for a select box to update a another select box, i.e. you want to wire the action to use the selected value when rendering the template.

Assumptions

Readers are assumed to be comfortable editing templates and have Zotonic Scomp knowledge.

How

We want to select an image from a page in 2 steps. First we select a page from the pages that have images. Then we select an image from the selected page.

This is how the main template will look:

```
<div class="form-group">
    {% wire id="pages" type="change" action={update
        target="media_list"
        template="_media_list.tpl"
    }%}
    <select id="pages" class="form-control">
        <option value="">--Select a page--</option>
        {% for page_id in m.search[{query
            hasobjectpredicate='depiction'
            sort='+rsc.pivot_title'
        }] %}
            <option value="{{ page_id }}">
                {{ m.rsc[page_id].title }}
            </option>
        {% endfor %}
    </select>
</div>
<div id="media_list">
    <!- contents will be updated -->
</div>
```

When an item is selected (by the change event), element with id "media_list" will be replaced by `_media_list.tpl` which looks like this:

```
{% with q.triggervalue as page_id %}
<div class="form-group">
    <select class="form-control">
        <option value="">--Select an image--</option>
        {% for media_id in m.rsc[page_id].media %}
            <option value="{{media_id}}">
                {{ m.rsc[media_id].title }}
            </option>
        {% endfor %}
    </select>
</div>
{% endwith %}
```

This template uses the q.triggervalue returned from the postback of the wire event, and it contains the value of the selected option.

We can go one step further to show the selected image. `_media_list.tpl` also gets a wire action:

```
{% with q.triggervalue as page_id %}
<div class="form-group">
    {% wire id="images" type="change" action={update
        target="image"
        template="_image.tpl"
    }%}
    <select class="form-control" id="images">
        <option value="">--Select an image--</option>
        {% for media_id in m.rsc[page_id].media %}
            <option value="{{media_id}}">
                {{ m.rsc[media_id].title }}
            </option>
        {% endfor %}
    </select>
</div>
<div id="image">
    <!- contents will be updated -->
</div>
{% endwith %}
```

And we show the image using `_image.tpl`:

```
{% with m.rsc[q.triggervalue].medium as medium %}
<div class="form-group">
    {% image medium width=300 %}
</div>
{% endwith %}
```

Admin cookbook

Creating a custom widget on the edit page

Why

For an imaginary webshop edit page, we want to add 2 more data fields: the affiliate url and a note about free shipping.

Assumptions

Readers are expected to have experience with Zotonic templates. For reference, look at the Zotonic admin template directory `modules/mod_admin/templates/`.

Custom widget

Content widgets are created by creating a file `_admin_edit_content.my_cat.tpl`, where `my_cat` is the category name - `webshop` in our case.

Create a file `_admin_edit_content.webshop.tpl`:

```
{% extends "admin_edit_widget_std.tpl" %}
```

Extending will also give us a couple of default properties like `id` and `is_editable`.

We can also override `admin_edit_widget_i18n.tpl` if fields need to get translated.

Add these block definitions:

```
{% block widget_title %}
    {_ Webshop Properties _}
{% endblock %}

{% block widget_show_minimized %}false{% endblock %}
{% block widget_id %}edit-webshop-properties{% endblock %}

{% block widget_content %}
    <fieldset class="form-horizontal">
        <div class="form-group row">
            <label class="control-label col-md-3">{_ Affiliate URL _}</label>
            <div class="col-md-9">
                <input type="text"
                    name="affiliate_url"
                    class="form-control"
                    value="{{ id.affiliate_url }}"
                    {% if not is_editable %}disabled="disabled"{% endif %}
                />
            </div>
        </div>
        <div class="form-group row">
            <label class="control-label col-md-3">{_ Free shipping note _}</label>
            <div class="col-md-9">
                <input type="text"
                    name="free_shipping_note"
                    class="form-control"
                    value="{{ id.free_shipping_note }}"
                    {% if not is_editable %}disabled="disabled"{% endif %}
                />
            </div>
        </div>
    </fieldset>
{% endblock %}
```

When loading a webshop edit page, the widget should now appear below the Basics block.

To customize the edit page further, see *Customizing the layout of the admin edit page* (page 233).

Word of caution

Admin templates will evolve over time. Custom changes you make now may not work after an update. Use these guidelines at your own risk.

Customizing the layout of the admin edit page

Why

After having created a custom widget (see *Creating a custom widget on the edit page* (page 232)), we want to hide widgets that we don't need.

Assumptions

Readers are expected to have experience with Zotonic templates. For reference, look at the Zotonic admin template directory `modules/mod_admin/templates/`.

Page structure

To hide default widgets from the page, we need to change the template that loads these widgets.

The edit page widgets are grouped into 2 main areas, main and sidebar. These groups are created by the files `_admin_edit_main_parts.tpl` and `_admin_edit_sidebar_parts.tpl`.

Content parts

We won't change the default Zotonic edit template; instead we will override it by adding the category to the filename.

Create a file `_admin_edit_main_parts.webshop.tpl` and give it the contents:

```
{% all catinclude "_admin_edit_basics.tpl" id is_editable=is_editable␣
↪languages=languages %}
{% all catinclude "_admin_edit_content.tpl" id is_editable=is_editable␣
↪languages=languages %}
{% include "_admin_edit_content_advanced.tpl" %}
```

This template will now load the Basic widget (Title, Summary, Short title), all content widgets (for now only `_admin_edit_content.webshop.tpl`), and the default Advanced widget.

Sidebar

For the sidebar area you can follow the same procedure. Of course you need to keep the submit buttons, so always include `_admin_edit_content_publish.tpl`.

Category and instance

To show a block on a category edit page, but not on an instance of that category, check for `r.is_a.meta`. To show the block "Features" on the category page we write:

```
{% if r.is_a.meta %}
{% include "_admin_edit_meta_features.tpl" %}
{% endif %}
```

Word of caution

Admin templates will evolve over time. Custom changes you make now may not work after an update. Use these guidelines at your own risk.

Customizing the style of an admin page

How to make style customizations to admin pages.

Assumptions

Readers are expected to have experience with Zotonic templates.

How

Creating a custom style for admin pages is best done using a module:

- In your site, create the directory `modules` if it doesn't exist.
- Create a directory for your skin, for instance `mod_skin_admin`.
- Populate the module directory with the module `.erl` file, sub directories `lib/css/` and `templates/` (see also: *Structure* (page 40)).
- Put your custom CSS file (f.i. `skin-admin.css`) in `lib/css/`.
- Create a template file `_html_head_admin.tpl` with the contents `{% lib "css/skin-admin.css" %}`.
- Enable the module in admin.

Storing date/time fields

Some interesting tidbits about saving/updating a date/time field of a rsc.

Why

The purpose of this guide is to explain how Zotonic stores date-time data so that you can work with it in your own modules.

Assumptions

Readers are expected to be interested in authoring Zotonic Modules. To be successful you should be fairly experienced with Erlang programming.

How

When updating a *resource* by passing a proplist to `m_rsc:update/3`, Zotonic has a way of looking for specific name patterns in the proplist that tells it what items are date/time fields and which ones are not. This allows Zotonic to split those off the proplists and use the name to derive information about the date, such as expected format, if the default time should be 00:00 or 23:59, and the actual field name that should be updated.

Here is an example format: `dt:ymd:0:date_end`

Literally this means ...

- the characters `dt:`, signifying the date-time marker
- followed by some year, month, date pattern or whatever combination
- followed by ":"
- followed by 1 or 0 to designate default 00:00 or 23:59

- followed by ":"
- followed by the field name to by updated.

Have a look at `_admin_edit_date.tpl` in the default site templates tosee this and other formatting options.

For example passing `[{"dt:ymd:0:date_end", "2011-04-05"}]` to `m_rsc:update/3` would update the date_end field.

Here is an example event callback:

```
event({submit, {edit_details, [{id, Id}]}, _TriggerId, _TargetId}, Context) ->
    End_date = z_context:get_q("dt:ymd:0:date_end", Context),
    Props1 = [{"dt:ymd:0:date_end", End_date}],
    case m_rsc:update(Id, Props1, Context) of
                {ok, _} ->
                   z_render:wire({reload, []}, Context);
                {error, _} ->
                    z_render:growl_error("Could not update record.", Context)
    end.
```

Of course the input field of your form could be named whatever you wish, but when you assign it to the proplist you would use the naming convention above.

Internals

`m_rsc:update` eventually hands off to `m_rsc_update:update`. In `m_rsc_upate`, zotonic passes the RscProps list to `recombine_dates/3` where it does a pattern match and list accumulator process.

Here is a description of the recombine_dates process:

```
recombine_dates([], Dates, Acc) ->
    % When our Props list is empty, return the list of
    % Dates we created and Acc containing the list of
    % non-date tuples.
    {Dates, Acc};
recombine_dates([{"dt:"++K,V}|T], Dates, Acc) ->
    % if the head of the list is a tuple with
    % {"dt:" plus some additional string with a value,
    % assign K to the rest of the sting and V to the
    % passed value.
    [Part, End, Name] = string:tokens(K, ":"),
    % With those values in hand, split K on ":" and
    % assign the tokens to Part, End, Name
    Dates1 = recombine_date(Part, End, Name, V, Dates),
    % Send our date 'packet' off for some more processing
    % and assign the return to
    % Date1 so we can start over again.
    recombine_dates(T, Dates1, Acc);
    % Call our function again withthe tail of our list
recombine_dates([H|T], Dates, Acc) ->
    % If the head of the list doesn't match "{"dt:"....}
    % we just pop it onto the head of the accumulator
    recombine_dates(T, Dates, [H|Acc]).
```

There is more to this process, but this is all you need to get started.

Admin template specific things

Common markup in admin templates.

Linking to edit pages

Dispatches to edit pages is done by `admin_edit_rsc`.

Linking:

```
{% url admin_edit_rsc id=my_id %}
```

Redirecting:

```
{% button text="edit" action={redirect dispatch="admin_edit_rsc" id=my_id} %}
```

Automatically add new users to a user group

Why

When you create a person, you usually need to add it to a user group as well. You may want to automate this, in particular if you need to differentiate the user group based on the person's category.

Note: If you merely want to change the default user group for *all* users, you can do so through a *configuration parameter* (page 312) and you don't need the solution below.

Solution

You can set the user group by adding a `hasusergroup` property to the resource. In your site's main `.erl` file, add a `rsc_update` observer:

```
-export([
    observe_rsc_update/3
]).

observe_rsc_update(#rsc_update{action = insert, id = Id}, {Modified, Props} = Acc,␣
↪Context) ->
    %% Where 'vip' is a subcategory of 'person'
    case m_rsc:is_a(Id, vip, Context) of
        false ->
            %% Do nothing
            Acc;
        true ->
            %% Add hasusergroup property
            {true, Props ++ [{hasusergroup, m_rsc:rid(acl_user_group_vips,␣
↪Context)}]}
    end;
observe_rsc_update(#rsc_update{}, Acc, _Context) ->
    %% Fall through
    Acc.
```

Shell cookbook

Activate/deactivate modules

Rescuing a dysfunctional site from the Zotonic shell.

Why

Sometimes it happens that disabled or enabled a module by accident and your whole site is now dysfunctional. You can easily rescue your site, from the Erlang command line.

How

First launch the Zotonic EShell:

```
~/zotonic/bin/zotonic shell
```

On the Zotonic EShell: Deactivate a module:

```
z_module_manager:deactivate(mod_modulename, z:c(yoursitename)).
```

On the Zotonic EShell: Activate a module:

```
z_module_manager:activate(mod_modulename, z:c(yoursitename)).
```

On the Zotonic EShell: Restart a module:

```
z_module_manager:restart(mod_modulename, z:c(yoursitename)).
```

Where *mod_modulename* is the name of your module and *yoursitename* is the name of your site.

Check Your Hostname

If you can't launch the shell chances are that your hostname is not a fully-qualified domain name (FQDN). This complicates things since Erlang is picky about allowing connections to unqualified nodes. For example, the hostname should be `myprimarysite.example.com` rather than something random and local like `coolserver`.

Filter and convert characters

Applying Erlang Binary syntax to get fast character manipulation.

- *Author: Lloyd R. Prentice*
- *Co-Author: Andreas Stenius*

Why

Erlang bit syntax is extraordinarily powerful and well worth learning.

Review documenation here: http://www.erlang.org/doc/programming_examples/bit_syntax.html

Here's an experiment. Follow along in your handy Erlang shell:

```
1> A = "The cat on the mat".
"The cat on the mat"
2> B = <<"The cat on the mat">>.
<<"The cat on the mat">>
3> io:format(A,[]).
The cat on the matok
4> io:format(B,[]).
The cat on the matok
So, big deal. What's the difference?
```

Memory consumption, that's what. A is represented internally as a list. Each character is a membory cell with a character and a pointer to the next cell. The list format is a memory hog. List represention of text strings is both bane and blessing for Erlang programmers. B, on the other hand, is mapped in memory as a contiguous array of characters, thus consuming much less memory.

For details on working with Erlang Binaries, look here: http://www.erlang.org/doc/man/binary.html

When you want to filter and convert characters in an Erlang Binary, however, things get dicey. It's all there in the Erlang documentation, but examples are few and far between.

The purpose of this particular recipe is to prepare text for conversion from a *.txt file to *.tex, a file marked up for input to the truly wonderful open source typesetting program Latex.

In our case, we don't have control of what gets typed into the source text file. So we need to both filter for unwanted characters and sequences as well as convert characters and sequences into LateX markeup.

What does this have to do with Zotonic?

You'll just have to wait and see.

Assumptions

All you'll need here is a text editor to copy and paste an Erlang module and an Erlang shell.

A passing understanding of Erlang list comprehensions will also be useful. Find more here: http://www.erlang.org/doc/programming_examples/list_comprehensions.html

How

Study latexFilter/1 below:

```
-module(filterz).
-export([latexFilter/1]).

%% Binary filters
%% with many thanks to Andreas Stenius
latexFilter(B) ->
   %% Delete control characters and extended ASCII
    B1 = << <<C>> || <<C>> <= B, (C == 10) or (C == 13) or (C >= 31),
(C =< 127) >>,
   %% Filter binary for conversion to *.tex format
   %% NOTE: Partially tested, but needs more
   F = [{"\r","\n"},                          % Convert returns to new lines
        {"\n *","\n"},                        % Delete spaces following newline
        {"^\\s*", ""},                        % Delete all whitespace
characters at beginning of manuscript
        {"^\n*", ""},                         % Delete new lines at
beginning of manuscript
        {"\n+$","\n"},                        % Delete excess new lines
at end of manuscript
        {"  +"," "},                         % Delete successive spaces
        {"\n{3,}","\n\\\\bigskip\n"},         % Convert three or more
newlines to Latex bigskip
        {"[&$#%~^{}]", "\\\\&"},              % Escape reserved Latex character
        {"<i>", "\\\\emph{"},                 % Convert HTML tag to Latex tag
        {"</i>", "}"},                        % Convert HTML tag to Latex tag
        {"\"(.*\")","``\\1"},                 % Convert opening double quotes (")␣
↪to Latex conventions (``)
        {" '"," `\\1"}],                      % Convert opening single quotes (')␣
↪to Latex conventions (`)
   B2 = lists:foldl(fun({Pattern, Replacement}, Subject) ->
                        re:replace(Subject, Pattern, Replacement,
```

```
                                    [global, {return, binary}]) end,
                        B1, F),
    {ok, B2}.
```

The first thing that happens in latexFiler/1 is an Erlang binary comprehension to delete pesky control and extended ASCII characters:

```
B1 = << <<C>> || <<C>> <= B, (C == 10) or (C == 13) or (C >= 31), (C =< 127) >>,
```

Note that the syntax is very similar to list comprehensions. As you can see, a single binary comprehension can chug out a lot of work. Define a few binaries in your Erlang shell and play around with the conditionals. You'll catch on pretty quick.

For our purposes, we need to also search and replace a bunch of substrings. For this we've enlisted `lists:foldl/3`, a powerful list function that just happens to work with binaries. See: http://www.erlang.org/doc/man/lists.html#append-1

Expect a headache when you study this function. It's not that easy to understand.

The `fold` function exists in two versions, `foldl` and `foldr` (for left and right, described shortly). It takes a list, and calls a function for each item in the list, along with a accumulator, or state. The function can operate on the item and state, producing a new state for the next round. The fold function also takes an initial state to use for the first item. The left and right mentioned previously determines in which order the list is traversed. So foldl moves through the list from left to right, i.e. takes the head off of the list for each iteration moving towards the tail; whereas the foldr starts with the tail and moves towards the head, i.e. from right to left. Regular expressions can also get hairy, but they're invaluable if you're working with text strings. For more information:

- http://www.troubleshooters.com/codecorn/littperl/perlreg.htm
- http://www.erlang.org/doc/man/re.html

Erlang tab completion

Get quicker access to Zotonic code on the shell.

Contributed by: Maas-Maarten Zeeman

Why

When you are working on the shell chances are you are looking to be fast. A way to be faster is to leverage the EShell's tab completion.

Assumptions

Readers are assumed to be comfortable using the EShell to run Erlang code interactively. This feature this guide describes only works with Zotonic modules only from `zotonic debug`. With the `zotonic shell` command, the Zotonic modules need to be explicitly loaded to work with tab-completion.

How

The Erlang shell has tab-completion. IIt is very handy for using in Zotonic, especially because of Zotonic's naming scheme. Just type:

```
1> z_<tab>
```

And you get a list of zotonic library modules. Type:

```
2> mod_<tab>, type m_<tab> a list of models.
```

You get the idea. The really nice thing is that it works for function names too.

Debugging db (query) issues

Techniques for finding root cause when queries are involved.

Why

When you face unexpected behavior as a result of some database query (z_db:q et al), you either have to hunt down the queries and re-run them by hand, which is really time consuming and painful, or, which this cookbook will detail, have postgresql log the queries for you.

The principal documentation and with more details, may be found here: http://www.postgresql.org/docs/8.4/static/runtime-config-logging.html

Assumptions

A working system where you want to inspect the db queries being run.

How

Edit your `postgresql.conf` file, enabling `logstatement = 'all'`. This file has a lot of options commented out, with comments which you may review. Options of interest in this scenario all begin with "log" (or similar).

By having `log_destination = 'stderr'` and `logging_collector = on`, you can capture your logging output to file.

Reset a user's password

Emergency password reset when you can't get into the admin interface.

Why

Sometimes it happens that you want to reset an user's password from the Erlang shell.

Assumptions

Readers are expected to be familiar with the EShell for running Erlang code interactively.

How

You can do this from the Erlang shell without using the /admin or the reset-password mail/dialog.

First go to the Erlang shell:

```
marc$ ./bin/zotonic shell
```

And then from the Erlang command prompt:

```
(zotonic@host)1> m_identity:set_username_pw(1234, "username", "password", z_
↪acl:sudo(z:c(yoursitename))).
```

Where *1234* is the id of your user (this must be an integer), `yoursitename` is the name of your site.

Troubleshooting

If you get the error:

```
{error, admin_password_cannot_be_set}
```

That means you are trying to change the password for the admin (user 1). The admin password is not set in the database: you need to define your admin password in the site's config file, use the property `admin_password`. For more info on this, see *Anatomy of a site* (page 11).

Restore/upgrade content db from backup

Contributed by: Scott Finnie

How to protect your Zotonic content and bring it back in case of disaster.

Why

Restore / upgrade the content for a site from a database dump.

Assumptions

You have a dump of the content to restore as a `.sql` file, which was created by *mod_backup* (page 286).

How

Dumps from postgres include both database structure (tables, sequences, constraints, etc.) as well as content. Therefore it's not possible to simply import the dump directly into an existing Zotonic database populated with content. The target database must be empty before the dump can be imported. There are (at least) 2 ways to do this:

1. Drop contents of the target db and import the dump file;
2. Create a new database, import contents, rename old database to something temporary, rename new database to match site, and finally delete the old database.

Option 2 seems more involved but is safer and quicker: it's non-destructive. The old db remains intact until after the new one is activated. If anything goes wrong, you can fall back to the original.

It also means less downtime. The site can stay live on the old db while the new db is being created. Downtime is restricted to the rename operations (which are quick and db size independent).

For small databases the downtime difference will be minimal. However the safety is appealing. Hence this cookbook uses the second approach.

Instructions

These assume the following naming conventions:

- Your Zotonic site is named yoursite.

- The postgres dump file to load is named zotonic-dump.sql.
- Zotonic is installed in ~zotonic/zotonic

Replace these assumptions as appropriate in all commands below. Ensure the db dump file is available on target machine (see pre-requisites).

Create a new database in postgres:

```
zotonic:~$ sudo su -l postgres
postgres:~$ ~Zotonic/zotonic/bin/zotonic createdb yoursite_new
CREATE DATABASE
GRANT
You are now connected to database "zotonic_yoursite_new".
CREATE LANGUAGE
postgres:~$
```

Import the dump file into the newly created db:

```
postgres:~$ psql -U zotonic -d zotonic_yoursite_new -f /path/to/zotonic-dump.sql
[...lots of SQL statements...]
postgres:~$
```

Note: the final sql commands shown will likely say 'no privileges could be revoked for "public"'. You can ignore this.

Stop Zotonic (if running):

```
postgres:~$ ~zotonic/zotonic/bin/zotonic stop
Stopping zotonic zotonic001@hostname
Stop:'zotonic001@hostname'
postgres:~$
```

Rename the databases:

```
postgres:~$ psql
postgres=# ALTER DATABASE zotonic_yoursite RENAME TO zotonic_yoursite_old;
ALTER DATABASE
postgres=# ALTER DATABASE zotonic_yoursite_new RENAME TO zotonic_yoursite;
ALTER DATABASE
postgres=# \q
postgres:~$ exit
zotonic:~$
```

Restart Zotonic:

```
zotonic:~$ ~/zotonic/bin/zotonic start
```

Browse to your site & test it's now serving updated content.

(Optional) Drop the old database:

```
zotonic:~$ sudo -u postgres psql
postgres=# DROP DATABASE zotonic_yoursite_old;
DROP DATABASE
postgres=# \q
zotonic:~$
```

Just enough...

These Cookbook items each represent a stage in some Zotonic users' journeys to understand the workings of Erlang and related technologies in general.

Just enough Erlang/OTP and rebar, part 1

Zotonic source code have you scratching your head? Learn Rebar first.

Created Aug 8, 2011 by Lloyd R. Prentice

Why

Rebar is a relatively new set of Erlang/OTP development tools. Rebar makes it easier to develop and maintain Erlang/OTP applications and releases.

Zotonic is built on Erlang/OTP. An understanding of Erlang/OTP conventions is essential if you wish to read Zotonic source code, develop Zotonic modules, or contribute code or patches to Zotonic.

As yet, Zotonic is not maintained under Rebar (external dependencies in *deps* are). But developing a skeletal Erlang/OTP application under Rebar is a great way to dip your feet into Erlang/OTP development.

By following each step in this tutorial carefully, and referring back to the many excellent on-line Erlang documentation resources, you will accelerate your progress up the daunting Erlang/OTP learning curve. And more, you'll learn how to read and understand Zotonic source code while you're at it.

In this tutorial we'll use rebar to create, compile, and test two Erlang applications. One will include a simple gen_server.

In Part II, we'll generate documentation, run eunit tests, and create a release that can be copied and run on a suitable host system.

Assumptions

You have Erlang R14B or later installed on your system. You have Internet access and basic Bash command line skills. File editing tasks refer to vim. But you can use your code editor of choice.

This tutorial was tested on Ubuntu 11.04.

How

How can I install and learn Rebar?

Create a root directory for experimentation. Let's call it "learn.":

```
$ mkdir learn
$ cd learn
```

Download the rebar binary

In the shell:

```
learn$ git clone https://github.com/basho/rebar.git rebar-src
learn$ cd rebar-src/
rebar-src$ ./bootstrap
rebar$ cd ..
learn$ cp rebar-src/rebar .
learn$ chmod u+x rebar
```

How can I create an application

In the shell:

```
learn$ ./rebar create-app appid=zzz
learn$ ls
>> rebar rebar-src src
```

Note that Rebar has created a directory named *src*:

```
learn $ ls src
>> zzz_app.erl zzz.app.src zzz_sup.erl
```

In *src*, Rebar has created three Erlang modules. Open them up and take a look in your favorite code editor:

Learn more about Erlang applications: http://www.erlang.org/doc/design_principles/applications.html http://www.erlang.org/doc/man/application.html

How can I add a gen_server template to my new application?

In the shell:

```
learn$ ./rebar create template=simplesrv srvid=zzz_srv
learn$ ls src
>> zzz_app.erl zzz.app.src zzz_srv.erl zzz_sup.erl
```

Open up `zzz_svr.erl` and look it over. For more info, study these `gen_gerver` resources:

http://www.erlang.org/doc/design_principles/gen_server_concepts.html http://www.erlang.org/doc/man/gen_server.html

How can I make my new gen_server do something?

In `src/zzz_srv.erl`, add two functions to the API `-export` directive as follows:

```
-export([start_link/0, say_hello/0, stop/0]).
```

Add the say_hello/0 function as follows:

```
say_hello() ->
  gen_server:call(?MODULE, hello).
```

Replace handle_call(_Request, _From, State):

```
handle_call(hello, _From, State) ->
  io:format("Hello from zzz_srv!~n", []),
  {reply, ok, State};
handle_call(_Request, _From, State) ->
  Reply = ok,
  {reply, Reply, State}.
```

Add the stop/0 function:

```
stop() ->
  gen_server:cast(?MODULE, stop).
```

Add before handle_cast(_Msg, State), put:

```
handle_cast(stop, State) ->
  {stop, normal, State};
```

NOTE: If your gen_server is under supervision, there's a better way to stop your server. See:

Section 2.6 of gen_server Concepts - Stopping: http://www.erlang.org/doc/design_principles/gen_server_concepts.html

You could compile this code with Rebar now, but let's defer.

To really get the hang, let's create TWO applications. We'll put them under a new directory, *apps/*:

```
learn$ mkdir apps
learn$ mkdir apps/zzz
learn$ mkdir apps/zzz_lib
learn$ ls apps
>> zzz zzz_lib
learn$ mv src apps/zzz/
learn$ ls apps/zzz
>> src
```

Now we'll create the zzz_lib application:

```
learn$ ./rebar create-app appid=zzz_lib
learn$ ls
>> apps rebar rebar-src src
```

And let's make it do something:

```
learn$ cd src
```

Create and save a module called `hello.erl` that does something:

```
-module(hello).
-export([hello/0]).
hello() ->
  io:format("Hello from zzz_lib!~n", []).
```

Back in the shell move the `src` directory to `apps/zzz_lib`:

```
src$ cd ..
learn$ mv src apps/zzz_lib/
```

How can I compile these two applications?

First, we need to create a `rebar.config` file in our project home directory. Create the file, add the following directive and save:

```
{sub_dirs, ["apps/zzz", "apps/zzz/src", "apps/zzz_lib", "apps/zzz_lib/src" ] }.
```

Back in the shell:

```
learn$ ls
>> apps rebar rebar-src rebar.config
```

Now compile:

```
learn$ ./rebar compile
```

If you see the following, pat yourself on the back:

```
==> zzz (compile)
Compiled src/zzz_app.erl
Compiled src/zzz_sup.erl
Compiled src/zzz_srv.erl
```

```
==> src (compile)
==> zzz_lib (compile)
Compiled src/hello.erl
Compiled src/zzz_lib_app.erl
Compiled src/zzz_lib_sup.erl
==> src (compile)
==> learn (compile)
```

Check out the ebin directories:

```
learn$ ls apps/zzz/ebin
>> zzz.app zzz_app.beam zzz_srv.beam zzz_sup.beam
learn$ ls apps/zzz_lib/ebin
>> hello.beam zzz_lib.app zzz_lib_app.beam zzz_lib_sup.beam
```

you're now ready to rock and roll!!

How can I test?

Start the Erlang shell:

```
learn$ erl -pa apps/*/ebin
1> zzz_srv:start_link().
{ok,<0.33.0>}
2> zzz_srv:say_hello().
Hello from zzz_srv!
ok
3> zzz_srv:stop().
ok
4> hello:hello().
Hello from zzz_lib!
ok
```

Troubleshooting

I got an error when I compiled. What now?

make sure your `rebar.config` directive, as shown above, is correct.

Make sure you have this directory structure:

```
learn$ tree
.
apps
|   - zzz
|   |   - ebin
|   |   - src
|   |       - zzz_app.erl
|   |       - zzz.app.src
|   |       - zzz_srv.erl
|   |       - zzz_sup.erl
|   - zzz_lib
|   |   - ebin
|       - src
|           - hello.erl
|           - zzz_lib_app.erl
|           - zzz_lib.app.src
|           - zzz_lib_sup.erl
- rebar
- rebar.config
```

Fix any source code errors, and recompile:

```
learn$ ./rebar compile
```

What you've learned

You've now had a good soak in basic Erlang/OTP conventions and Erlang. You can install Rebar, create Erlang/OTP applications, and compile them. You've also created a simple gen_server.

Where to go from here

Study the online and printed Erlang documentation upside and sideways. Skim to see what's there, then reread everytime you have a problem. You'll be an Erlang/OTP wizard before you know it.

References on the web

Getting Started: https://github.com/basho/rebar/wiki/Getting-started

Damn Technology: http://damntechnology.blogspot.com/

How to create, build, and run an Erlang OTP application using Rebar: http://skeptomai.com/?p=56#sec-3

Commands: https://github.com/basho/rebar/wiki/Rebar-commands

Erlang App. Management with Rebar: http://erlang-as-is.blogspot.com/2011/04/erlang-app-management-with-rebar-alan.html

Dizzy Smith – Building Erlang Applications with Rebar: http://ontwik.com/erlang/dizzy-smith-building-erlang-applications-with-rebar/

Rebar Demo using ibrowse: http://vimeo.com/8311407

rebar / rebar.config.sample: https://github.com/basho/rebar/blob/master/rebar.config.sample?source=cc

Books

Programming Erlang: Software for a Concurrent World: http://www.amazon.com/Programming-Erlang-Software-Concurrent-World/dp/193435600X

Erlang Programming: http://www.amazon.com/ERLANG-Programming-Francesco-Cesarini/dp/0596518188/ref=pd_sim_b_1

Erlang and OTP in Action: http://www.amazon.com/Erlang-OTP-Action-Martin-Logan/dp/1933988789/ref=pd_sim_b_1

Just enough Erlang/OTP and rebar, part 2

Building a gen_server to front the library and generating documentation.

Created Aug 17, 2011 by Lloyd R. Prentice

WHY

If you worked through the Cookbook item *Just enough Erlang/OTP and rebar, part 1* (page 244), you have created two Erlang applications under management of Rebar. You've used Rebar to create a gen_server template in the first application and added several functions. You've also created a library application with one function.

In this tutorial we'll call our library function from our gen_server and generate documentation.

ASSUMPTIONS

You have Erlang R14B or later installed on your system. You have basic Bash command line skills. File editing tasks refer to vim. But you can use your code editor of choice.

You've successfully compiled the zzz and zzz_lib applications from Part I under Rebar.

This tutorial was tested on Ubuntu 11.04.

HOW

How can I call a library function from zzz_lib in a zzz module?

In the shell:

```
learn$ cd apps/zzz/src
```

Edit handle_call/2 in zzz_srv.erl as follows:

```
handle_call(hello, _From, State) ->
  hello:hello(),
  io:format("~nHello from zzz_srv!~n", []),
  {reply, ok, State};
```

While you're at it, check your handle_cast/2 code. It should look like this:

```
stop() ->
  gen_server:cast(?MODULE, stop).

%% callbacks
handle_cast(stop, State) ->
  {stop, normal, State};

handle_cast(_Msg, State) ->
  {noreply, State}.
```

Now edit and save init/1 in learn/apps/zzz/src/zzz_sup.erl:

```
init([]) ->
   {ok, {{one_for_one, 1, 60}, [?CHILD(zzz_srv, worker)]}}.
```

Now recompile:

```
learn$ ./rebar clean compile
```

And test:

```
learn$ erl -pa apps/*/ebin
...
1> zzz_sup:start_link().
{ok,<0.33.0>}
```

Note that you may see a different PID on your system. Try some functions:

```
2> zzz_srv:say_hello().
Hello from zzz_lib!
Hello from zzz_srv!
ok
```

Both our library function and zzz_svr have performed as we'd hoped.

While developing this tutorial, I introduced a bug in `zzz_svr:handle_cast/2`. Once it started, I couldn't stop. This was a good excuse to introduce a valuable debugging tool.

How can I kill a running process?

In the Erlang shell, start *pman*, the process manager:

```
3> pman:start().
<0.37.0>
```

You should see a graphic window open with a list of running processes. Find `zzz_srv` and `zzz_sup` under Name. Click on zz_sup to highlight. Now pull down the Trace menu item and click on *kill*. Note that you've not only killed zzz_sup, but zzz_srv as well. Of course. zzz_sup supervises zzz_srv. When it dies, so does zzz_srv.

You can confirm:

```
4> zzz_srv:say_hello().
** exception exit: {noproc,{gen_server,call,[zzz_srv,hello]}}
   in function  exit/1
      called as exit({noproc,{gen_server,call,[zzz_srv,hello]}})
   in call from gen_server:call/2
```

You assassin, you!

But, no worry, you can start it back up again:

```
5> zzz_srv:start_link().
{ok,<0.48.0>}
```

Now let's test zzz_srv:stop().:

```
6) zzz_srv:stop().
ok
```

Seemed to worked. But...:

```
7> zzz_srv:say_hello().
Hello from zzz_lib!
Hello from zzz_srv!
ok
```

It turns out, zzz_sup started it up again. Try it again:

```
8> zzz_srv:stop().
** exception exit: shutdown
```

Wait! If you look at pman, you'll see that zzz_sup also died. What's going on here?

If you look at zzz_sup.erl in learn/apps/zzz/src, you'll note that init/1 allows only one restart, the value following "one_for_one.":

```
init([]) ->
   {ok, {{one_for_one, 1, 60}, [?CHILD(zzz_srv, worker)]}}.
```

For more details, checkout the Supervisor Behaviour: http://www.erlang.org/doc/design_principles/sup_princ.html

How can I generate documentation?

Add comments to learn/apps/zzz/src/zzz_srv.erl as follows:

```
%% @spec say_hello() -> none
%% @doc display greeting
say_hello() ->
```

And to learn/apps/zzz_lib/src/hello.erl:

```
%% @spec hello() -> none
%% @doc Library test function
hello() ->
```

Now execute:

```
learn$ ./rebar clean compile
...
learn$ ./rebar doc
==> zzz (doc)
==> zzz_lib (doc)
```

Check out your new doc directories:

```
learn$ ls apps/zzz
doc  ebin  src

learn$ ls apps/zzz_lib
doc  ebin  src
```

Now bring up your browser and point to file:

```
file:///home/learn/apps/zzz/doc/index.html file:///home/learn/apps/zzz_lib/doc/
↪index.html
```

TROUBLESHOOTING

I got an error when I compiled. What now?

make sure rebar.config in ../learn looks like this:

```
{sub_dirs,
        ["apps/zzz",
         "apps/zzz/src",
         "apps/zzz_lib",
         "apps/zzz_lib/src"
        ]
}.
```

Make sure you have this directory structure:

```
learn$ tree
.
apps
|   - zzz
|   |   - ebin
|   |   - src
|   |       - zzz_app.erl
|   |       - zzz.app.src
|   |       - zzz_srv.erl
|   |       - zzz_sup.erl
|   - zzz_lib
|   |   - ebin
|       - src
|           - hello.erl
|           - zzz_lib_app.erl
|           - zzz_lib.app.src
|           - zzz_lib_sup.erl
- rebar
- rebar.config
```

WHAT YOU'VE LEARNED

You've now had a good soak in basic Erlang/OTP conventions and Erlang. You can install Rebar, create an Erlang/OTP application, compile it and create documentation.

WHERE TO GO FROM HERE

Study the online and printed Erlang documentation upside and sideways. Skim to see what's there, then reread everytime you have a problem. You'll be an Erlang/OTP wizard before you know it.

Now, dive into Zotonic source code. It should be much easier to follow.

REFERENCES

Getting Started: https://github.com/basho/rebar/wiki/Getting-started

Damn Technology: http://damntechnology.blogspot.com/

How to create, build, and run an Erlang OTP application using Rebar: http://skeptomai.com/?p=56#sec-3

Commands: https://github.com/basho/rebar/wiki/Rebar-commands

Erlang App. Management with Rebar: http://erlang-as-is.blogspot.com/2011/04/erlang-app-management-with-rebar-alan.html

Dizzy Smith – Building Erlang Applications with Rebar: http://ontwik.com/erlang/dizzy-smith-building-erlang-applications-with-rebar/

Rebar Demo using ibrowse: http://vimeo.com/8311407

rebar / rebar.config.sample: https://github.com/basho/rebar/blob/master/rebar.config.sample?source=cc

Just enough Postgres

Understand the primary data-store of Zotonic.

Why

Data persistence in Zotonic is provided by PostgreSQL, a mature feature-rich open-source relational database.

Since Zotonic provides both a data model and wrapper around PostgreSQL queries, the Zotonic user is substantially insulated from routine Postgres operation. But now and again, for issues ranging from installation, backup/restore, and debugging, familiarity with a few PostgreSQL commands can save considerable time.

Sadly, PostgreSQL user documentation is abundant, but not as well organized as one might hope.

Assumptions

These commands have been tested on Debian Squeeze and Ubuntu 11.04.

Familiarity with SQL and relational databases advised.

NOTE: The string "VERSION," as used below, refers to your PostgreSQL version: E.g. 8.3 for Debian Lenny, 8.4 for Debian Squeeze and Ubuntu 11.04

How

Where can I find PostgreSQL documentation?

If PostreSQL is installed: `/usr/share/doc/postgresql-common/README.Debian.gz`.

For on-line documentation: http://www.postgresql.org/docs/VERSION/static/

For instance, in Debian Lenny, look for: http://www.postgresql.org/docs/8.3/static/

Is PostgreSQL installed?

In a shell:

```
ls /usr/lib | grep postgresql
```

You should see:

    postgresql

What version?

In the shell:

```
ls -l /usr/lib/postgresql/
```

You should see:

```
drwxr-xr-x 4 root root 4096 2011-06-23 14:13 VERSION
```

How can I install PostgreSQL?

In the shell:

```
apt-get update
apt-get install postgresql postgresql-client
```

Or, from source: http://www.postgresql.org/docs/8.4/static/install-short.html

Where are Postresql files located?

Configuration files: `/etc/postgresql/[version]/[cluster]/` Binaries: `/usr/lib/postgresql/[version]` Data files: `/var/lib/postgresql/[version]/[cluster]`

Is the PostgreSQL server running?

In the shell:

```
/etc/init.d/postgresql status
```

You should see something like:

```
Running clusters: 8.4/main
```

How can I stop the PostgreSQ server?

In the shell:

```
/etc/init.d/postgresql stop
```

You should see something like:

```
* Stopping PostgreSQL 8.4 database server      [ OK  ]
```

How can I start the Postgres server?

In the shell:

```
/etc/init.d/postgresql start
```

You should see something like:

```
* Starting PostgreSQL 8.4 database server       [ OK ]
```

How can I restart the PostgreSQL server?

In the shell:

```
/etc/init.d/postgresql restart
```

You should see something like:

```
* Restarting PostgreSQL 8.4 database server
```

How can I switch to database 'zotonic_blog' in psql?

In the shell:

```
zotonic@host $ psql
zotonic=# \c zotonic_blog
```

You should now be on psql for the zotonic_blog database:

```
You are now connected to database "zotonic_pcc".
zotonic_blog=#
```

How can I enter the PostgreSQL interactive terminal?

In the shell:

```
psql
```

You should now be on the interactive terminal:

```
psql (8.4.8)
Type "help" for help.

postgres=#
```

How can I list databases?

From psql:

```
\l
```

Or directly from the Zotonic User's shell:

```
psql -l
```

You should see a list of databases like the following:

```
                                  List of databases
     Name      |  Owner   | Encoding |  Collation  |    Ctype    |   Access␣
↪privileges
---------------+----------+----------+-------------+-------------+----------------------
 postgres      | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 template0     | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 | =c/postgres
                                                                 : postgres=CTc/
↪postgres
 template1     | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 | =c/postgres
                                                                 : postgres=CTc/
↪postgres
```

```
 zotonic      | zotonic  | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 zotonic_blog | zotonic  | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
(5 rows)
```

How can I see if a database exists?

In the shell:

```
psql test
```

If the database doesn't exist:

```
psql: FATAL: database "test" does not exist
```

If the database exists, you'll see something like:

```
psql (8.4.8)
Type "help" for help.

test=>
```

How can I tell if the database for site 'blog' exists in the local postgres installation?

In the shell:

```
psql -l | grep blog
```

You should see something like:

```
zotonic_blog | zotonic  | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
```

How do I list the relations (tables, views , sequences) in a database?

In the shell:

```
psql zotonic_blog
zotonic_blog=# \d
```

You should see something like:

```
                    List of relations
 Schema |               Name                |   Type   |  Owner
--------+-----------------------------------+----------+---------
 public | category                          | table    | zotonic
 public | comment                           | table    | zotonic
 public | comment_id_seq                    | sequence | zotonic
 public | config                            | table    | zotonic
 public | config_id_seq                     | sequence | zotonic
 public | edge                              | table    | zotonic
{...etc. }
```

If psql displays this in a pager (prompt is a :) you can escape by hitting q.

How can I create a table in a database?

NOTE: Many fine books and tutorials are available to help you learn SQL, the standard query language for relational databases. See references below.

The follow queries are for illustration only:

```
postgres=# CREATE TABLE books (
postgres(# title text NOT NULL);
CREATE TABLE
```

How to add a column to a table:

```
postgres=# ALTER TABLE books
postgres-# ADD author text NOT NULL;
ALTER TABLE
```

How to examine the structure of a table:

```
postgres=# \d books
   Table "public.books"
 Column | Type | Modifiers
--------+------+-----------
 title  | text | not null
 author | text | not null
```

How to insert a record into a table:

```
postgres=# INSERT INTO books ( title, author )
postgres-# VALUES ('Programming Erlang', 'Joe Armstrong');
INSERT 0 1
```

How to examine records in a table:

```
postgres=# SELECT * FROM books;

       title        |    author
--------------------+---------------
 Programming Erlang | Joe Armstrong
(1 row)
```

How to select a record from a table:

```
postgres=# SELECT title FROM books
postgres-# WHERE author = 'Joe Armstrong';
       title
--------------------
 Programming Erlang
(1 row)
```

How to create a database user:

```
postgres=# CREATE USER myuser WITH PASSWORD 'userpassword';
CREATE ROLE
```

How to create a database:

```
postgres=# CREATE DATABASE testdb WITH OWNER = myuser ENCODING = 'UTF8';
CREATE DATABASE
postgres=# GRANT ALL ON DATABASE testdb TO myuser;
GRANT
```

How to initialize a database:

http://www.postgresql.org/docs/8.4/static/app-initdb.html

How can I back-up a database:

— Method 1: Use Backing up your site.

— Method 2: Dump can be created on the source machine with the following command (replace zotonic_blog with your site's db name):

```
pg_dump zotonic_blog > zotonic_blog.sql
```

How to delete a database named 'test' and all its contents:

```
pg_dump test > test.sql
dropdb test
```

How can I restore the contents of a database from backup

See *Restore/upgrade content db from backup* (page 242)

Zotonic Conveniences that avoid direct Postgres interaction

How can I create a database for my first Zotonic?:

```
zotonic createdb blog
zotonic addsite -d zotonic_blog blog
```

How can I create a database for an additional Zotonic site?:

```
zotonic createdb blog
zotonic addsite -d zotonic_blog blog
```

Notice the pattern ;)

How can I open the Zotonic shell?

In the terminal:

```
zotonic shell
```

How can I select records from the Zotonic shell?

In the zotonic shell:

```
1> m_rsc:get(page_home, z:c(blog)).
[{category_id,104},
 {created,{{2011,6,8},{22,21,55}}},
 {creator_id,1},
 {id,313},
 {is_authoritative,true},
 {is_featured,false},
 {is_protected,false},
 {is_published,true},
 {modified,{{2011,6,8},{22,21,55}}},
 {modifier_id,1},
 {name,<<"page_home">>},
 {page_path,<<"/">>},
 {publication_end,{{9999,8,17},{12,0,0}}},
 {publication_start,{{2011,6,8},{22,21,55}}},
 {slug,<<"home">>},
 {uri,undefined},
 {version,1},
 {visible_for,0},
 {title,<<"Home">>},
 {summary,<<"Welcome to your blog!">>},
 {managed_props,[{title,<<"Home">>},
                 {summary,<<"Welcome "...>>},
                 {page_path,<<"/">>}]},
 {installed_by,z_install_defaultdata}]
```

Troubleshooting

Pay GREAT attention to permissions. Your tables and sequences should be owned by the user specified in the site's config file. GRANT may not be enough. So, if you see Zotonic trying to recreate tables or if Zotonic fails

with a 3D000 error (database object doesn't exist) even if you are positive already exist, it means your permissions are wrong.

Problem:

You try to get an psql shell:

```
psql
```

And it refuses to work:

```
psql: FATAL:  Ident authentication failed for user "postgres"
```

Solution:

You need to configure `pg_hba.conf`

Note: For maximum security, correct configuration of pg_hba.conf is essential.

See *Enabling trust-authentication in PostgreSQL* (page 489) in this manual, or look at the Postgresql docs:

http://www.postgresql.org/docs/8.4/interactive/client-authentication.html http://www.postgresql.org/docs/8.4/interactive/auth-pg-hba-conf.html

Problem:

In postgres, you get the following:

```
testdb=> CREATE USER testdb WITH PASSWORD 'testdb';
ERROR:  permission denied to create role
```

Solution:

You need to create a database user. Retry as the Postgres superuser:

```
sudo su postgres psql
```

And it will work:

```
postgres=# CREATE USER testdb  WITH PASSWORD 'testb';
CREATE ROLE
```

Problem:

In the shell:

```
cd /etc/postgresql
```

Outputs:: bash: cd: /etc/postgresql: No such file or directory

Solution:

This is evidently a bug in certain Debian Lenny installs when /etc/postgresql is inadvertently deleted. Uninstalling postgresql-client (`apt-get --purge remove postgresql-client`) is supposed to fix it. But it won't if the system has an older version of udev.

See: http://bugs.debian.org/cgi-bin/bugreport.cgi?bug=517389

Need updated version of udev

Problem:

Erratic performance of database

Solution:

Examine PostgreSQL installation files. Expect trouble if, by happenstance, you have more than one instance of PostgreSQL server running. You may have to back-up your data, uninstall all PostgreSQL files and reinstall.

Note: On some Lenny installations `apt-get --purge remove postgresql` will *NOT* remove all configuration files. And, `apt-get install posgtresql` will not replace missing a missing `/etc/postgresql` directory and files.

Resources

Howto: Debian / Ubuntu Linux Install PostgreSQL Database Server http://www.cyberciti.biz/faq/linux-installing-postgresql-database-server/

psql: FATAL: Ident authentication failed for user "username" Error and Solution http://www.cyberciti.biz/faq/psql-fatal-ident-authentication-failed-for-user/

PostgreSQL for Beginners http://www.postgresqlforbeginners.com/2010/11/interacting-with-postgresql-psql.html

PostgreSQL 8.4.8 Documentation http://www.postgresql.org/docs/8.4/static/index.html http://www.postgresql.org/docs/8.4/static/reference-client.html

Howto Backup PostgreSQL Databases Server With pg_dump command http://www.cyberciti.biz/tips/howto-backup-postgresql-databases.html

How To Use pg_dump and pg_restore with Postgres Plus® Tutorial for Linux® http://www.enterprisedb.com/resources-community/tutorials-quickstarts/linux/how-use-pgdump-and-pgrestore-postgres-plus-tutorial-

An almost idiot's guide to Install and Upgrade to PostgreSQL 8.4 with Yum http://www.postgresonline.com/journal/archives/144-An-almost-idiots-guide-to-Install-and-Upgrade-to-PostgreSQL-8.4-with-Yum.html

postgresql clustering and Debian http://www.progsoc.org/~wildfire/notes/postgresql-cluster.html

Books

Momjian, Bruce, PostgreSQL: Introduction and Concepts, 2001, Addison-Wesley, Upper Saddle River, NJ, 462 pp

Worsley, John C. and Joshua D. Drake, Practical PostgreSQL, 2002, O'Reilly & Associates, Inc., Sebastopol, CA, 618 pp

Just Enough Regular Expressions (in Erlang)

Learn how to manipulate string data with the `re` module.

Lloyd R. Prentice August 31, 2011

WHY

The Erlang standard library re provides a powerful suite of functions to excute regular expressions to find, replace, and manipulate substrings within a string or Erlang binary.

Re functions are particularly useful for form validation, decomposing and modifying urls, e-mail addresses, and other common web elements represented as strings or binaries.

They are found throughout Zotonic source files.

See: http://www.erlang.org/doc/man/re.html

ASSUMPTIONS

You have a current version of Erlang installed on your system.

Examples have been tested on Ubuntu 11.04.

HOW

Bring up an Erlang terminal:

```
$ erl
Erlang R14B03 (erts-5.8.4) [source] [64-bit] [smp:3:3] [rq:3] [async-threads:0]␣
↪[kernel-poll:false]

Eshell V5.8.4  (abort with ^G)
1>
```

and follow along with the following examples. Modify and re-execute each example until you feel comfortable with what's going on.

What is a regular expression?

A regular expression is a pattern that is matched against a subject string from left to right.

The power of regular expressions comes from the ability to include alternatives and repetitions in the pattern. These are encoded in the pattern by the use of metacharacters.

What is a pattern?

Most characters stand for themselves in a pattern.

The pattern "ick", for instance, would match the first occurence of "ick" in the string "The quick brown fox."

We'll use `re:run/2` to illustrate. This function checks a string agains a regular expression: `run(Subject, RE) -> {match, Captured} | nomatch`.

Example:

```
6> re:run("The quick brown fox.","ick").
{match,[{6,3}]}
```

The atom "match" is self-explanatory. The tuple {6,3} in the list provides the start position and length of the pattern in the subject string.

```
7> re:run("The brown fox.","ick").
nomatch
```

re:run/2 will also work with a binary:

```
8> re:run(<<"The quick brown fox.">>,"ick").
{match,[{6,3}]}
```

If we wish to find all instances of "ick" in a string, we need to use re:run/3, `run(Subject,RE,Options) -> {match, Captured} | match | nomatch`.

Example:

```
9> re:run("The sick quick brown fox.", "ick", [global]).
{match,[[{5,3}],[{11,3}]]}
```

For documentation of re:run/3 options, see http://www.erlang.org/doc/man/re.html

How can I replace a substring in a string?

Use `re:replace/3`: `replace(Subject, RE, Replacement) -> iodata() | unicode:charlist()`.

Example:

```
10> re:replace("The quick brown fox.", "brown", "red").
[<<"The quick ">>,<<"red">>|<<" fox.">>]
```

Hmmm... re:replace/3 returns an odd-looking binary, which is called an *iolist*: an efficient data structure, used like this to prevent copying of data in memory.

But, let's use re:replace/4 to provide an option: `replace(Subject, RE, Replacement, Options) -> iodata() | unicode:charlist()`:

```
11> re:replace("The quick brown fox.", "brown", "red", [{return, list}]).
"The quick red fox."
```

The `{return, list}` does the trick of returning a "regular" string. Erlang documentation is generally thorough, but often not that easy to follow. That's why I created this Cookbook item. I wanted to learn this stuff myself.

Regular expressions can deliver much much more, however, with shrewd use of metacharacters.

What is a metacharacter?

Metacharacters are interpreted in special ways. For instance, the metacharacter . matches the first instance of any character in a string except newline.

Example:

```
13> re:run("The quick brown fox.", ".").
{match,[{0,1}]}
```

You'd usually use . in a more elaborate pattern:

```
14> re:run("The quick brown fox.", "qu.").
{match,[{4,3}]}

15> re:run("The quack brown fox.", "qu.").
{match,[{4,3}]}
```

The metachacter ^ asserts start of string.

Examples:

```
16> re:run("The quack brown fox.", "^The").
{match,[{0,3}]}

17> re:run("The quack brown fox.", "^qua").
nomatch
```

Similarly, the metacharacter $ asserts the end of a line:

```
18> re:run("The quick brown fox is sick.", "ick.$").
{match,[{24,4}]}
```

The metacharacter * matches zero or more characters.

Examples:

```
19> re:run("The quick brown fox.", "i*").
{match,[{0,0}]}

20> re:run("The quick brown fox.", "T*").
{match,[{0,1}]}
```

```
21> re:run("TTTTThe quick brown fox.", "T*").
{match,[{0,5}]}
```

The metacharacter + matches one or more characters:

```
22> re:run("TTTTThe quick brown fox.", "z+").
nomatch

23> re:run("TTTTThe quick brown fox.", "T+").
{match,[{0,5}]}
```

The metacharacter | alternate patterns. Think of it as "or":

```
24> re:run("The quick brown fox.", "fox|pig").
{match,[{16,3}]}

25> re:run("The quick brown pig.", "fox|pig").
{match,[{16,3}]}
```

You can also match generic character types. s, for instance matches any whitespace character.

Examples:

```
26> re:run("The quick brown fox","\s",[global]).
{match,[[{3,1}],[{9,1}],[{15,1}]]}
```

How can I match non-printing characters?

See: Non-printing characters http://www.erlang.org/doc/man/re.html

Note that the metacharacters [and] have special meaning, they enclose "character classes." A character class is the set of characters in a character class match, if any found, one character in the subject string.

Examples:

```
24> re:run("The quick brown fox.", "[qui]").
{match,[{4,1}]}

25> re:run("The quick brown fox.", "[ui]").
{match,[{5,1}]}

26> re:run("The quick brown fox.", "[qui]", [global]).
{match,[[{4,1}],[{5,1}],[{6,1}]]}
```

You can combine characters, meta-characters, and other regular expression elements into extended patterns that can search, match, and replace nearly any substrings you can imagine.

Example:

```
27> re:run("E-mail: xyz@pdq.com", "[a-zA-Z0-9]+@[a-zA-Z0-9]+\.[a-z]{2,3}").
{match,[{8,11}]}
```

Note: DO NOT use this pattern in production. It needs more refinement and much more testing.

What other goodies does `re` offer?

`split(Subject, RE) -> SplitList` and `split(Subject, RE, Options) -> SplitList`.

Examples:

```
28> re:split("this/is/my/path","/").
[<<"this">>,<<"is">>,<<"my">>,<<"path">>]
```

If you wish to use a pattern multiple times and boost perfomance, you can compile it with `re:compile/1`.

Example:

```
29>  {_, P} = re:compile("[a-zA-Z0-9]+@[a-zA-Z0-9]+\.[a-z]{2,3}").
{ok,{re_pattern,0,0,
                <<69,82,67,80,164,0,0,0,0,0,0,0,5,0,0,0,0,0,0,0,0,64,
                  ...>>}}
30> re:run("E-mail: xyz@pdq.com", P).
{match,[{8,11}]}
```

How are regular expressions used in Zotonic source?

For one of many examples, look at `zotonic/src/markdown/get_url/1`:

```
get_url(String) ->
  HTTP_regex = "^(H|h)(T|t)(T|t)(P|p)(S|s)*://",
  case re:run(String, HTTP_regex) of
      nomatch    -> not_url;
      {match, _} -> get_url1(String, [])
  end.
```

Where can I go from here?

Study and experiment with all the metacharacters and other regular expression constructs in:

http://www.erlang.org/doc/man/re.html

Do further research on the web. Everytime you see an interesting regular expression, test it in re:run/2. You may well have to edit to get it to run on re:run/2. But if you understand the basics, it won't be difficult.

TROUBLESHOOTING

CAUTION: Complex regular expression patterns are hard to read and error prone. Break them down into short segments and test each segment. Then build them back up.

The hard part is confirming that your pattern will match all possible instances of the string segments you're interested in.

RESOURCES

http://www.erlang.org/doc/man/re.html http://langref.org/erlang/pattern-matching http://www.regular-expressions.info/examples.html

Just enough Erlang shell

An indispensible tool for both learning and programming Erlang.

Submitted by: LRP; July 30, 2011

WHY

You don't need to know Erlang or use the Erlang shell to create simple Zotonic websites. But for Zotonic application development and debugging, Erlang programming skills are essential.

Erlang is not that hard to learn. Several excellent books and any number of web tutorials will get you well underway.

The Erlang shell is an indispensible tool for both learning and programming Erlang.

The easiest way to learn how to use the Erlang shell is to fire it up and play.

This Cookbook recipe provides all you need to get started.

ASSUMPTIONS

You have a recent version of Erlang installed on your system.

HOW

What is the Erlang shell?

The Erlang shell lets you test Erlang expression sequences both locally and remotely. It also lets you work with Erlang records and manage Erlang jobs.

How can I enter the Erlang shell?

From your terminal command line:

```
$ erl
```

(and press enter). You'll see something like:

```
Erlang R13B03 (erts-5.7.4) [source] [64-bit] [smp:3:3] [rq:3] [async-threads:0]␣
↪[hipe] [kernel-poll:false] Eshell V5.7.4 (abort with ^G)
1>
```

`1>` is the Erlang shell command line. You are now able to execute Erlang expressions.

How can I get help?

help(). – list of shell functions How can I execute an expression?

Type now `now()`. and then hit ENTER.

NOTE: the period at the end of the expression is necessary. It terminates the Erlang expression sequence and initiates evaluation.

How can I edit a line?

- Ctrl+A – go to beginning of line
- Ctrl+E – go to end of line
- `li` <Tab> – tab completiion; for instance, li tab will return `lists:` * `li` <Tab> <Tab> – Erlang terms that contain the characters "li"

For more shell editing commands, refer to Section 1.3.1 in: http://www.erlang.org/documentation/doc-5.1/doc/getting_started/getting_started.html

How can I manage Erlang jobs?

- Ctrl+G – user switch command; job control options
- Ctrl+G h – help user switch options

How can I leave the shell?

`q().` – quits Erlang

Are there other ways to quit?

- halt(). – exits the Erlang runtime system
- Ctrl+C Ctrl+C – Display break options, then quit

BEWARE: You don't want to bring a running Erlang system down just to quit the shell. Use q() until you get the hang of stuff.

When I enter Ctl+C, I get a bunch of choices. What do they mean?

Refer to: http://www.erlang.org/doc/man/shell.html

REFERENCES

http://www.erlang.org/documentation/doc-5.1/doc/getting_started/getting_started.html

http://www.erlang.org/doc/man/shell.html

http://www.erlang.org/doc/getting_started/seq_prog.html

Erlang: Starting Out: http://learnyousomeerlang.com/starting-out

Just enough Webmachine

Getting a solid foundation for your Zotonic skillset.

WHY

Zotonic talks to the web through a wonderful Erlang-based http REST toolkit called *Webmachine*. Webmachine was generously contibuted to the open-source world by Basho of Cambridge, MA.

A basic understanding of Webmachine will provide a solid foundation for your Zotonic skillset.

This Cookbook recipe will give you hands-on experience with Webmachine, Webmachine resources, and how to modifify a Webmachine resource.

ASSUMPTIONS

You have git, a recent version of Erlang, and Zotonic installed on your system.

You have basic Linux shell and program editing competencies.

You have basic sequential Erlang skills and understand the structure of Erlang modules.

HOW

We'll install and compile Webmachine, create a simple demo application, and modify and execute a Webmachine resource. Finally, we'll take a brief look at how Zotonic integrates Webmachine.

NOTE: Your Zotonic install already includes a webmachine instance. But we'll install another instance so you can play.

What is Webmachine?

Webmachine handles well-formed http REST requests, gracefully handles all errors, and provides powerful debugging tools.

A Webmachine application is a set of *resources* written in Erlang, each of which is a set of functions over the state of the resource.

Note: in Zotonic's version of Webmachine, we renamed *resource* to *controller*, as we think that's a better name for those things. And in Zotonic, resources are "pages".

These functions give you a place to define the representations and other Web-relevant properties of your application's resources

How can I explore Webmachine?

Make sure that you have a working Erlang/OTP release, R12B5 or later.

Get the webmachine code:

```
git clone git://github.com/basho/webmachine
```

Build webmachine:

```
cd webmachine
make
```

Create, build, and start the skeleton resource:

```
./scripts/new_webmachine.sh mywebdemo /tmp
cd /tmp/mywebdemo
make
./start.sh
```

You should see a series of progress reports. Point a web browser at http://localhost:8000/

You should now see Hello, new world

How can I stop the demo?

Press ctl C, then press a

How can I examine the resource that produced the hello-world message?

In the shell:

```
$ cd /tmp/mywebdemo/src
```

Open mywebdemo_resource.erl in your favorite code editor.

How can I modify the resource to execute and display my own Erlang function?

Make the following changes mywebdemo_resource.erl just under the init/1 function:

```
earthman() ->
    "Earth man!".

to_html(ReqData, State) ->
    {"<html><body>Hello, " ++ earthman() ++ "<body></html>", ReqData, State}.
```

Save, and compile:

```
mywebdemo$ make
```

Now, from your web browser, refresh http://localhost:8000. You should see your change.

You'll find more Webmachine documentation at: http://webmachine.basho.com/docs.html. It'll take you pretty much wherever you want to go.

You now know enough about Webmachine to understand how Zotonic communicates through the web.

So how does Zotonic integrate Webmachine? Peek into your Zotonic *deps* directory:

```
$ cd /home/zotonic/zotonic/deps
```

You'll see the''webzmachine'' directory, which is Zotonic's version of webmachine it's contents will be quite familiar from your explorations above, although some implementation details differ.

REFERENCES

http://webmachine.basho.com/

CHAPTER 4

Best practices

Best practices as formulated by the Zotonic core team.

Best Practices

Creating sites

Initialize your site with a proper data model and some resources through `manage_schema`.

Template Best Practices and Pitfalls

This chapter lists some preferred solutions to common tasks and pitfalls you may encounter while developing with templates.

Variable Naming

Name your variables for what they represent. If you are searching for articles, name the search result variable `article` to make things clear.

In particular, if you are iterating over a list or other iterable variable called images, then your item variable should be named `image`. This makes generative templates easier to follow.

Use `id` instead of `m.rsc`

For the most frequently used model *m_rsc* (page 426), use a shortcut syntax: write `{{ id.title }}` instead of `{{ m.rsc[id].title }}`. So this:

```
<li>
  <h3>{{ m.rsc[id].title }}</h3>
  {% for image_id in m.rsc[id].o.depiction %}
  <figure>
      {% media image_id width=100 link=id %}
      {% if m.rsc[image_id].summary %}
          <p class="image-caption">{{ m.rsc[image_id].summary }}</p>
```

```
      {% endif %}
  </figure>
  {% endfor %}
</li>
```

becomes:

```
<li>
    <h3>{{ id.title }}</h3>
    {% for image_id in id.o.depiction %}
    <figure>
      {% media image_id width=100 link=id %}
      {% if image_id.summary %}
          <p class="image-caption">{{ image_id.summary }}</p>
      {% endif %}
    </figure>
  {% endfor %}
</li>
```

Pitfalls

Using 'm' as a template variable blocks model access

Avoid using the name `m` for a variable. It has a special meaning in templates for accessing models like *m_site* (page 433) and *m_rsc* (page 426) as *m.site* and *m.rsc*. Effectively. all Erlang modules with names starting with *m_* are made available in templates through the `m` variable.

It is particularly important since using m as a variable name will disable model module access for the entire scope within which that variable is defined. This can lead to very confusing template errors.

Using 'id' as a template variable blocks access to the current page

When rendering a page through *controller_page* (page 357), Zotonic sets the *id* variable based on the *resource* (page) being rendered. It is also conventionally used by dispatch rules to supply the id of the page to render.

Note, however, that there are legitmate cases for using id as a template variable. It is a good idea to reuse another template to render a section for one page and treating related content as the "current page" for that template by assigning id in a for loop or with context. Using id otherwise will likely confuse other developers trying to read your templates.

CHAPTER 5

Reference

A complete overview of all Zotonic components.

Reference

All the nitty gritty details when the big picture has settled.

Modules

Zotonic comes with a considerable number of modules included. This section documents each of them, explaining their uses and their relationships to other modules.

To learn more about how a module works and what it consists of, we refer you to the *Modules* (page 39) manual.

Looking for more modules?

Check out the Zotonic Module Index, an index with additional user-contributed modules which are not part of the core Zotonic distribution.

Module listing

mod_acl_adminonly

The default ACL module for providing access to the admin.

When enabled, each user that is created is allowed access to the admin interface, with full permissions.

This module is convenient when you just need a simple access model, usually with a single user who manages the whole site and knows what he is doing.

mod_acl_user_groups

This module adds rule-based access control.

- All resources (pages) are put into a content group.

- All users are member of zero or more user groups.
- Content groups are arranged in a hierarchy.
- User groups are arranged in a hierarchy.

ACL rules are defined between user groups and content groups. Each single rule gives some user group one or more permissions on some content group.

Managing user groups

By default, Zotonic has four user groups:

- anonymous (anonymous visitors of the website)
- members (logged in users of the website)
- editors (content editors)
- managers (manage users)

These user groups are arranged in a hierarchy, so that each group has the permissions its parent plus more. So, the permissions of users in the members group include those of anonymous users; and editors have the permissions of both anonymous users and members plus more.

To add or remove user groups, go to *Auth* > *User groups* in the admin.

Collaboration groups

Collaboration groups are a special type of user groups. They are most useful when you have groups of users that together collaborate on content. All content belongs to the group. Each collaboration group has one or more managers. So if you have groups of students working together and being supervised by teachers, you can define them as collaboration groups with the teachers as managers.

Managing content groups

By default, Zotonic has two content groups:

- default (all site content)
- system content (categories, predicates and user groups)

To add or remove content groups, go to *Structure* > *Content groups* in the admin. Just like user groups, content groups are ordered in a hierarchy. So the permissions that apply to the parent content group also apply to all of its children.

Defining access rules

You can add ACL rules in two ways:

1. in the admin web interface at `http://yoursite/admin/acl/rules`
2. in your site's or module's code; see *Managed rules* (page 274) below.

Let's start by defining rules in the web interface.

Content access rules

Each content access control rule grants some user group one or more permissions on some content group. So, for each rule you must specify the following properties:

- User group
- Content group
- Resource category (or 'all categories')
- Permissions: 'view', 'insert', 'update', 'delete', 'link', 'manage own'.

If you wish to narrow down the rule, you can select a single resource category it applies to. The default is 'all categories'.

The content group dropdown contains:

- all your *Managing content groups* (page 272)
- all your *Collaboration groups* (page 272)

The permissions include simple resource permissions that determine whether users in the group are allowed to view, insert, update or delete resources. The 'link' permission is about creating outgoing edges from resources in the content group to other resources.

Some rules may be greyed out and have a note saying 'This rule is managed by module . . . '. These are *Managed rules* (page 274) that you cannot edit in the web interface.

Collaboration group rules

Collaboration rules are special content access rules that apply to content in *collaboration groups* (page 272) only. Each rule applies to all collaboration groups.

Module access rules

Each module access rule grants some user group *use* permissions on some module. In the admin, go to *Auth* > *Access control rules* > *Modules* tab to edit them.

Deny rules

By default, rules grant some permissions. But sometimes you want to deny some permissions that are granted by another rule. For instance, if you have a rule that allows anonymous users to view all content groups, but you have a special content group 'Top secret' that you want to hide from anonymous users, add a rule to deny access:

Deny	ACL user group	Content group	Category	Permissions
	Anonymous	Top secret	All	View

Publishing rules

When you're editing rules, they are not effective immediately: you have to publish them first. Click the 'Publish' button to do so.

You can test out your rules before publishing them by clicking the 'Try rules. . . ' button.

Managed rules

Above you've seen how you can add rules through the web interface. Using *module versioning* (page 44), you can also write rules in your code. These rules are called 'managed rules' because they are defined in the code of modules, including your own *site module* (page 11).

While editing a simple set of ACL rules in the web interface is easier for end users, developers may prefer to manage more complex rules in code. Managed rules have two important advantages:

- they are equal between all environments (such as development, acceptance and production)
- when developing and deploying new features, ACL rules and code often belong together. By defining the rules in your code, you can commit and store them along with the feature code.

If you haven't yet done so, set up *module versioning* (page 44) in `yoursite.erl` or `mod_your_module.erl`. Then, in the `manage_schema/2` function, add an `acl_rules` section under the `data` property in the `#datamodel{}` record:

```
%% yoursite.erl
-module(yoursite).
-mod_schema(1).

-export([
    manage_schema/2
]).

%% .... more code here...

manage_schema(install, Context) ->
    #datamodel{
        %% your resources...
        data = [
            {acl_rules, [
                %% A resource ACL rule is defined as {rsc, Properties}.
                {rsc, [
                    {acl_user_group_id, acl_user_group_members},
                    {actions, [view, link]},
                    {is_owner, true},
                    {category_id, person}
                ]},

                %% A module rule is defined as {module, Properties}
                {module, [
                        {acl_user_group_id, acl_user_group_editors},
                        {actions, [use]},
                        {module, mod_ginger_base}
                    ]
                }
            ]
        ]
    }.

manage_schema({upgrade, 2}, Context) ->
    %% code to upgrade from 1 to 2
    ok;
```

Compile the code and restart your module to load the managed rules. They will be added and immediately *published* (page 273).

The set of rules defined in `manage_schema/2` is *declarative* and *complete*. That is to say, you declare the full set of rules that you wish to define. Any changes or deletions that you make to the rules in your code, will propagate to the site's rules. To protect the set's completeness, managed rules cannot be altered in the web interface.

Exporting and importing rules

To back up your rules, go to *Auth* > *Access control rules* and click the 'Export edit rules' button. The backup will include the full hierarchies of all user and content groups.

You can import a previous backup by clicking the 'Import edit rules...' button.

mod_admin

Todo

Finish documentation

Extending the admin menu

See *m_admin_menu* (page 413) on how to extend the admin menu.

Extending the admin edit page

There are several special templates names that will be automatically included into the */admin/edit/xxx* page from when you create these specially named templates.

`_admin_edit_basics.tpl` Will be automatically included into main (left) div (at top).

`_admin_edit_content.tpl` Will be automatically included into main (left) div (at bottom).

`_admin_edit_sidebar.tpl` Will be automatically included into right sidebar (near middle/bottom).

These templates are included using the *all catinclude* (page 441) tag; so if you need something in the sidebar just for persons, create a `_admin_edit_sidebar.person.tpl` file in your project.

Overriding TinyMCE options

If you need to override TinyMCE options; adding plugins, or setting other settings; you can create an `_admin_tinymce_overrides_js.tpl` file which can contain extra settings for the TinyMCE editors in the admin.

The template must contain JavasSript which modifies the *tinyInit* variable just before the editor is started. For example, to tweak the "paste" options you can put the following in the template:

```
tinyInit.theme_advanced_blockformats = "p,h1,h2"
tinyInit.paste_auto_cleanup_on_paste = true;
tinyInit.paste_remove_styles = true;
tinyInit.paste_remove_styles_if_webkit = true;
tinyInit.paste_strip_class_attributes = true;
tinyInit.paste_text_sticky = true;
tinyInit.paste_text_sticky_default = true;
```

TinyMCE Zotonic options

Zotonic provides extra init options:

`z_insert_dialog_enabled` Set this to false to prevent the insert media dialog from showing. Default *true*.

`z_properties_dialog_enabled` Set this to false to prevent the media properties dialog from showing. Default *true*.

Writing admin widget templates

This section contains examples of templates to create widgets for the /admin. Each of these examples extends basic several widget templates from mod_admin. To write your own you need to drop example content and fill holes in these example widgets.

You can use them as basis for your site admin-related tasks.

_admin_dashboard_example.tpl Very simple example widget for admin dashboard. Contains blocks for title and body. Look at /admin to see several dashboard widgets (latest events, pages, media, etc).

_admin_widget_std.tpl Sligthly more complex widget example. Same templates are used into /admin/edit/N for main content and sidebar widgets. These widgets do not provide any localization abilities. Also note that there are several special widget names:

_admin_widget_i18n.tpl Complex widget example. Is used to edit localized rsc properties. It will be rendered as tabs. See /admin/edit/N top left to see the tabs. If mod_translation disabled, then i18n-widgets are displayed same as _admin_widget_std.tpl.

Making an admin widget conditionally visible

To make an entire admin widget visible or not, depending on some condition that you want to calculate inside the widget's code, you can use the *widget_wrapper* block (which sits around the entire widget) in combination with the *inherit* (page 452) tag, wrapping that with a condition.

For instance, *mod_backup* (page 286) uses this technique to display the import/export sidebar widget. Excerpt from mod_backup's *_admin_edit_sidebar.tpl*:

```
{# Make the widget conditional, based on the config value mod_backup.admin_panel #}
{% block widget_wrapper %}
    {% if m.config.mod_backup.admin_panel.value %}
        {% inherit %}
    {% endif %}
{% endblock %}
```

In this example, when the condition is true, the wrapper is rendered normally (content is inherited from the extended template); when false, the wrapper block is overridden by new (but void) content.

See also:

template-admin_edit_widget_i18n, *inherit* (page 452)

Extending the admin overview page

The overview page at `admin/overview` shows a table with links to all pages. It can be filtered and sorted. This extension deals with the view when a category filter has been selected.

The goal is to add more specific information that helps to distinguish pages.

The *Category* column, second from the left, can be extended to carry category-specific information. As an example, pages of category Event show the start date for each event, grayed out when the event is in the past. Something like this:

Title	Starts	Created on
Great event	2016-08-27 00:00:00	25 Aug 2015, 22:19
Past event	2014-01-02 00:00:00	01 Jan 2014, 13:10

Instead of dates, the information can be anything - from color coding labels to location data, the number of comments or the completeness state of product descriptions.

Setting up templates

To make it work we are using 3 templates (where *category_name* is the lowercase name of your category):

`_admin_overview_list.category_name.tpl`

Overrides the overview with a `field` variable for our custom sort. If we are using *an existing resource property* (page 426) such as `date_start`, we write:

```
{% include "_admin_overview_list.tpl"
  field="rsc.pivot_date_start"
%}
```

`_admin_sort_header.category_name.tpl`

The sort header caption. For a non-sortable header, just write the caption as text. For a sortable header, include the sort functionality in `_admin_sort_header.tpl` and pass the caption as variable:

```
{% include "_admin_sort_header.tpl"
    caption="My caption"
    type="date"
%}
```

`type="date"` indicates that sorting should start descending, from new to old.

`_admin_overview_list_data.category_name.tpl`

Contains the category-specific information. This is a freeform Zotonic template. The Events example checks if the end date is in the past:

```
{% if id.date_end|in_past %}
    <span class="text-muted">{{ id.date_start|timesince }}</span>
{% else %}
    <span>{{ id.date_start|timesince }}</span>
{% endif %}
```

Custom sort properties

To sort on values that are not stored in the default Zotonic resources, you will need to create a *custom pivot* (page 205). This will create an additional database table with the values to sort on.

Let's take the (unlikely) example where we want to display the summary of each page (and sort on it as well). The summary data is not stored in an easily accessible way (at least for sorting), so we need to add 2 pivot methods to our Erlang module:

```
init(Context) ->
    z_pivot_rsc:define_custom_pivot(?MODULE, [{summary, "binary"}], Context),
    ok.

observe_custom_pivot({custom_pivot, Id}, Context) ->
    case m_rsc:p(Id, summary, Context) of
        {trans, [{en, Summary}]} ->
            {?MODULE, [{summary, Summary}]};
        _ -> none
    end.
```

For this example we are just grabbing the English text, and assume that other translations exist.

The `field` name in `_admin_overview_list.category_name.tpl` now just needs to contain the pivot column name:

```
{% include "_admin_overview_list.tpl"
    field="summary"
%}
```

And the sort header template `_admin_sort_header.category_name.tpl` adds the custom pivot variable:

```
{% include "_admin_sort_header.tpl"
    caption="Summary"
    custompivot="my_module"
%}
```

Resource meta *features*

Resources in the meta category can have 'features': certain resource properties (usually in the form of checkboxes) that decide what to show or hide on certain pages in the admin. To use this, create a `_admin_features.category.tpl` in your module.

For instance, the module *mod_geomap* defines the following `_admin_features.category.tpl` to create an extra checkbox so that per category can be defined whether or not the geodata box should be shown:

```
<div class="controls">
        <label class="checkbox">
        <input value="1" type="checkbox"
               name="feature_show_geodata"
               {% if id.feature_show_geodata|if_undefined:`true` %}checked{% endif
↪%}
               />
        {_ Show geo data on edit page _}
    </label>
</div>
```

And on the edit page there is this check to conditionally include the geodata box:

```
{% if id.category_id.feature_show_geodata|if_undefined:`true` %}
```

The `if_undefined` is used so that the default value can be true when the checkbox has never been touched.

Configuration keys

For the admin there are two configuration keys: `mod_admin.rsc_dialog_tabs` and `mod_admin.rsc_dialog_is_published`.

The `mod_admin.rsc_dialog_tabs` key defines which tabs are shown in the new resource, media-upload, and image-link dialogs. Per defauls these dialogs show all the possible tabs, with this configurarion key it is possible to change that.

The tabs are: `find,new,upload,url,embed,oembed,depiction`

The `depiction` is used for the TinyMCE image-link dialog; it shows all media connected using the `depiction` predicate.

The `mod_admin.rsc_dialog_is_published` defines the default *is_published* state for new resources being mad in the *new* tab. Setting this key to *1* will check the *is_published* checkbox.

See also:

if_undefined (page 409)

mod_admin_category

Add support for editing *Categories* (page 5) in the admin, by presenting an editable category tree at `http://yoursite.com/admin/category`.

Note: This module requires the presence of *mod_menu* (page 301) for the required JavaScript files which make up the menu editor.

ACL permissions

The following *ACL permissions* (page 38) are required:

- to view the page, *use permission* (page 273) on the 'mod_admin_category' module
- to view the list of categories, *view permissions* (page 273) on category 'category'
- to edit and re-order the categories, *edit permissions* (page 273) on category 'category'.

Todo

Add more documentation

mod_admin_config

Add support for editing the site's configuration values, as accessed through *m_config* (page 415).

The page in the admin is a list of every configuration module, key and textual value. Entries can be added, removed, and edited, if the user has the permission to do so.

When a value is large than 65 characters, it will be truncated in the admin config list view. To view the whole value, click the row.

See also:

m_config (page 415)

Todo

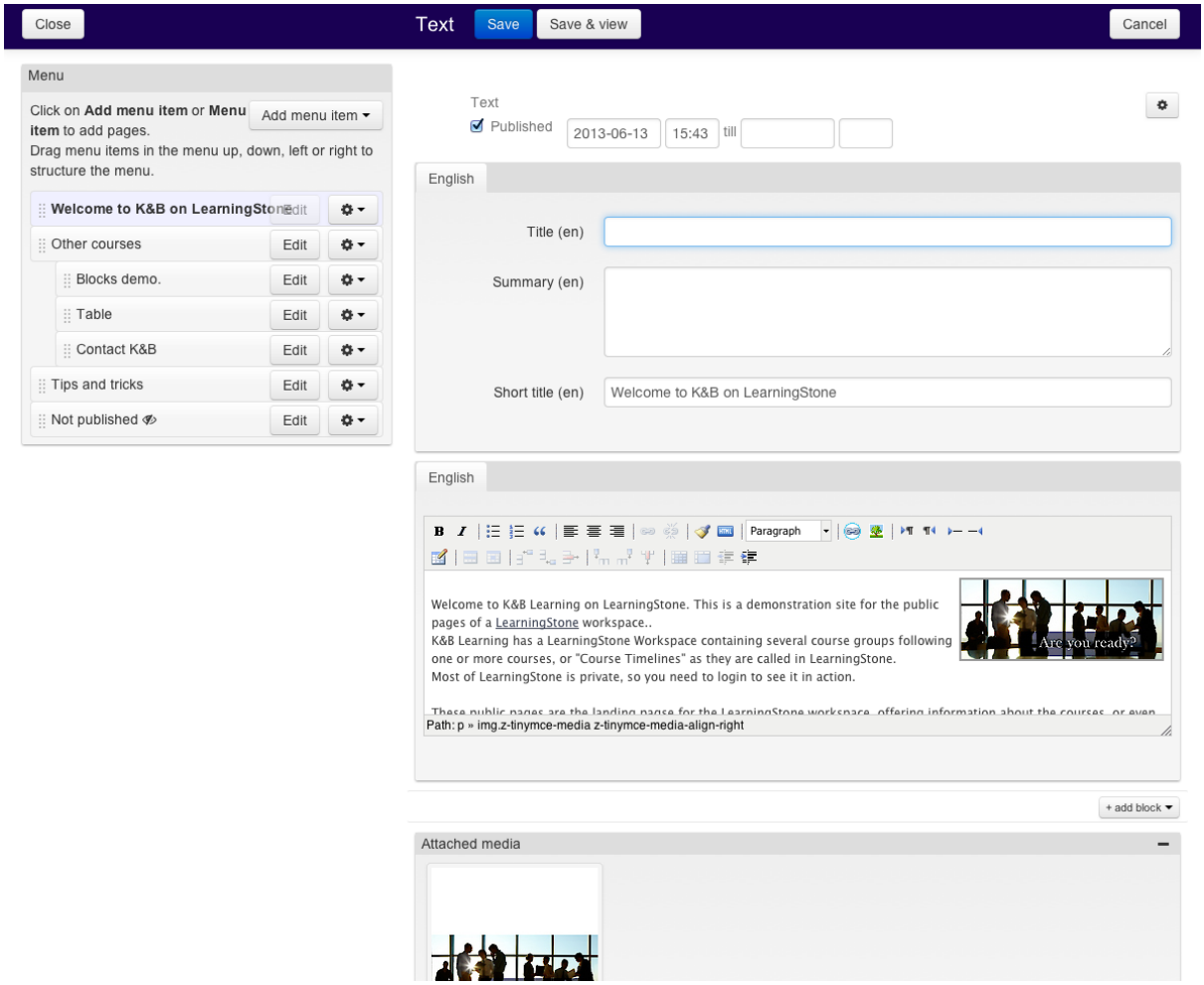
Add more documentation

mod_admin_frontend

Adds editing of resources, menu-trees and collections for non-admin users.

With many sites it is needed to let "normal" users edit content. For those users the `/admin` interface is overwhelming and also gives too many options that only administrators should be using.

For this use-case the mod_admin_frontend is created. It reuses some templates and parts of the normal admin, but in such a way that only the most needed options and fields are present. It also enables editing a menu-tree or collection, side-by-side with the content in the menu-tree.



Interface and dispatching

The interface is a one-page interface which uses postbacks to update the currently edited resource and forms. The editing history is maintained by using hashes in the url.

The URL of the edit page is generated by the dispatch rule `admin_frontend_edit`:

```
[
    {admin_frontend_edit, ["edit"], controller_page, [{acl, is_auth}, {template,
↪"page_admin_frontend_edit.tpl"}]},
    {admin_frontend_edit, ["edit", id], controller_page, [{acl_action, edit},
↪{template, {cat, "page_admin_frontend_edit.tpl"}}]}
].
```

The template uses the `menu_rsc` filter to find the contextual menu for the resource being edited. Per default the `menu_rsc` filter will fallback to the resource with the name `main_menu`. Hook into the `#menu_rsc{}` notification to change this behavior.

To edit a specific resource in the context of a menu that is non-standard for the resource, use the following code:

```
<a href="{% url admin_frontend_edit id=my_menu %}#edit_id={{ id }}">Click to edit {
↪{ id.title }}</a>
```

Customizing and templates

The `mod_admin_frontend` makes heavy use of `catinclude` to find the correct templates.

The column with the menu is rendered using `{% catinclude "_admin_menu_menu_view.tpl" tree_id ... %}`. The resource-column with the main editing form is rendered using `{% catinclude "_admin_frontend_edit.tpl" id ... %}`. This column is loaded lazily via a postback.

Check the templates in the

Customizing the menu column

Extra content can be added above or below the menu view by overruling the blocks `above_menu` and `below_menu`.

Customizing the main edit column

The main edit template `_admin_frontend_edit.tpl` provides the main edit-form, javascript initializations and fields for basic publication status editing.

The main edit block `edit_blocks` is defined as follows and can be overruled for specific categories:

```
{% block edit_blocks %}
    {% catinclude "_admin_edit_basics.tpl" id is_editable=is_editable␣
↪languages=languages %}

    {% if id.category_id.feature_show_address %}
        {% catinclude "_admin_edit_content_address.tpl" id is_editable=is_editable␣
↪languages=languages %}
    {% endif %}

    {% all catinclude "_admin_edit_content.tpl" id is_editable=is_editable␣
↪languages=languages %}

    {% if id.is_a.media or id.medium %}
        {% include "_admin_edit_content_media.tpl" %}
    {% endif %}

    {% catinclude "_admin_edit_body.tpl" id is_editable=is_editable␣
↪languages=languages %}
    {% catinclude "_admin_edit_blocks.tpl" id is_editable=is_editable␣
↪languages=languages %}
    {% catinclude "_admin_edit_depiction.tpl" id is_editable=is_editable␣
↪languages=languages %}
{% endblock %}
```

If any content needs to be added on top of the page, between the publication checkbox and the main edit fields, then overrule the block `meta_data_after`.

If you click on the *cog* icon on the right, then a meta data panel is shown (for access-control options and language settings). This panel can be extended with extra tabs using the blocks `meta_tabs` and `meta_panels`.

Note: This module is in active development. In the future live-editing and push-updates will be added. This might change the way the form and menu are handled.

See also:

mod_admin (page 275), *menu_rsc* (page 392)

mod_admin_identity

Provides identity management in the admin - for example the storage of usernames and passwords.

It provides a user interface where you can create new users for the admin system. There is also an option to send these new users an e-mail with their password.

Create new users

See the *User management* (page 6) chapter in the User Guide for more information about the user management interface.

Configure new user category

To configure the category that new users will be placed in, set the `mod_admin_identity.new_user_category` configuration parameter to that category's unique name.

Password Complexity Rules

By default Zotonic only enforces that your password is not blank. However, if you, your clients or your business require the enforcement of Password Complexity Rules Zotonic provides it.

Configuring Password Rules

After logging into the administration interface of your site, go to the *Config* section and click *Make a new config setting*. In the dialog box, enter `mod_admin_identity` for Module, `password_regex` for Key and your password rule regular expression for Value.

After saving, your password complexity rule will now be enforced on all future password changes.

Advice on Building a Password Regular Expression

A typical password_regex should start with ^.* and end with .*$. This allows everything by default and allows you to assert typical password rules like:

- must be at least 8 characters long (?=.{8,})
- must have at least one number (?=.*[0-9])
- must have at least one lower-case letter (?=.*[a-z])
- must have at least one upper-case letter (?=.*[A-Z])
- must have at least one special character (?=.*[@#$%^&+=])

Putting those rules all together gives the following password_regex:

```
^.*(?=.{8,})(?=.*[0-9])(?=.*[a-z])(?=.*[A-Z])(?=.*[@#$%^&+=]).*$
```

To understand the mechanics behind this regular expression see Password validation with regular expressions.

Migrating from an older password hashing scheme

When you migrate a legacy system to Zotonic, you might not want your users to re-enter their password before they can log in again.

By implementing the `identity_password_match` notification, you can have your legacy passwords stored in a custom hashed format, and notify the system that it needs to re-hash the password to the Zotonic-native format. The notification has the following fields:

```
-record(identity_password_match, {rsc_id, password, hash}).
```

Your migration script might have set the `username_pw` identity with a marker tuple which contains a password in MD5 format:

```
m_identity:set_by_type(AccountId, username_pw, Email, {hash, md5, MD5Password},␣
↪Context),
```

Now, in Zotonic when you want users to log on using this MD5 stored password, you implement `identity_password_match` and do the md5 check like this:

```
observe_identity_password_match(#identity_password_match{password=Password, hash=
↪{hash, md5, Hash}}, _Context) ->
  case binary_to_list(erlang:md5(Password)) =:= z_utils:hex_decode(Hash) of
      true ->
          {ok, rehash};
      false ->
          {error, password}
  end;
observe_identity_password_match(#identity_password_match{}, _Context) ->
  undefined. %% fall through
```

This checks the password against the old MD5 format. The `{ok, rehash}` return value indicates that the user's password hash will be updated by Zotonic, and as such, this method is only called once per user, as the next time the password is stored using Zotonic's internal hashing scheme.

mod_admin_merge

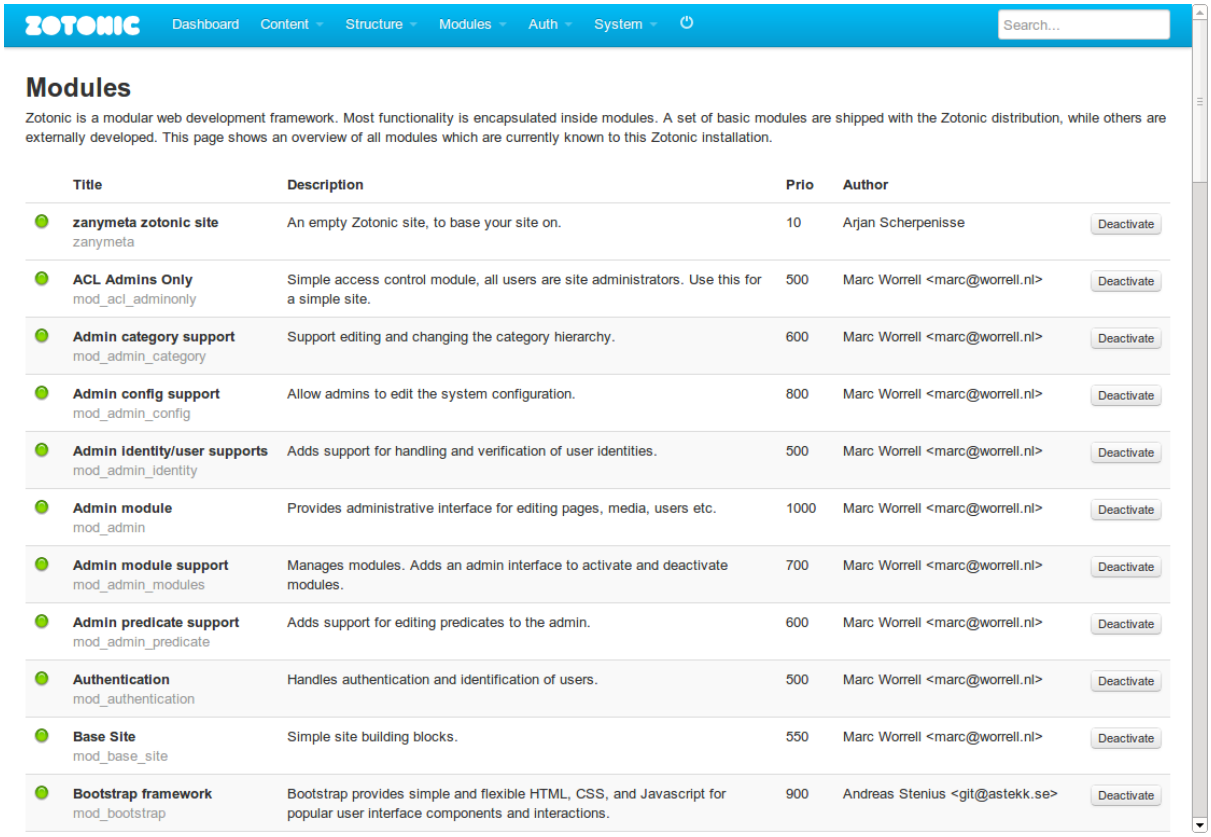
Todo

Not yet documented.

mod_admin_modules

Adds support in the admin for activating and deactivating *modules*.

The module overview in the admin presents a list of all modules, ordered by module status and priority. An activate/deactivate button allows the module to be (de)activated.



Module configuration dialog

When activate, some modules have a "configuration" button next to the activate/deactivate button. In that case, you can click the button to pop up a configuration dialog where you can set options which are required to configure the module.

To create such a dialog yourself, include a template called `_admin_configure_module.tpl` in your `templates/` folder in your module.

mod_admin_predicate

Add support for editing *predicates* (page 5) in the admin, by presenting a list of all defined predicates on `http://yoursite.com/admin/predicate`.

Predicates can be added, removed and edited, just like regular *resources*.

ACL permissions

The following *ACL permissions* (page 38) are required:

- to view the page, *use permission* (page 273) on the 'mod_admin_predicate' module
- to edit and delete predicates, *edit and delete permissions* (page 273) on category 'predicate'.

Todo

Add more documentation

mod_artwork

This module contains many useful icons and images.

Included CSS icons

lib/material-design/ Material Design Iconic Font.

lib/font-awesome-4/ Font Awesome 4: fontawesome.io.

lib/font-awesome/ Font Awesome 3.2.1.

How to use Material Design icons in your templates

- Include the CSS file: `{% lib "material-design/css/material-design-iconic-font.min.css" %}`
- Follow the examples
- Find the class names in the icons overview

How to use Font Awesome icons in your templates

- Include the CSS file: `{% lib "font-awesome-4/css/font-awesome.min.css" %}`
- Follow the icon examples
- Find the class names in the cheatsheet

Included images

lib/images/emotes/ A collection of emoticons. These are from the Tango Icon Library and in the Public Domain.

lib/images/flags/ A collection of country flags. They are coded using the two letter ISO-3166 code of the country. For example `lib/images/flags/flag-nl.png` refers to the Dutch flag. This collection is from Wikimedia Commons and in the Public Domain. The PNG files are created by the Open Icon Library.

lib/images/zotonic/ Some Zotonic logos, with space for more.

lib/images/mimeicons/ Zotonic can't generate a preview of all files. If it can't generate a preview then an icon representing the mime type of the file is used. This is a collection of images for some mime types. This collection is from the Tango Icon Library and in the Public Domain.

lib/images/noun/ A selection of icons from The Noun Project. There are many black&white icons in this directory. This collection is licensed with the Creative Commons Attribution 3.0 Unported (CC BY 3.0) License, though about half of the icons are in the Public Domain or licensed using CC 0. When you use an icon in your project, check the license on http://thenounproject.com/ and the proper attribution as described in http://thenounproject.com/using-symbols/

lib/images/social/round/ Round icons for various social sites. Think of Youtube, Twitter, Facebook etc. This collection is made by Veodesign (http://veodesign.com/) It is licensed under the Creative Commons Attribution Share-Alike 3.0 Unported License. This means you are not allowed to sell these icons and you need to properly attribute Veodesign.

lib/images/social/square/ Square icons for various social sites. This collection comes from the Open Icon Library (homemade) and is in the Public Domain. These icons are only 64x64 pixels (all others are 256x256).

How to use images in your templates

Most of the icons are in 256x256 PNG format. That is too large for normal usage. Best is to resize the images in your templates using the *image* (page 449).

For example, to display a 64x64 pixel image:

```
{% image "lib/images/social/round/email.png" width=64 %}
```

mod_atom

Support for representing resources as Atom XML.

Todo

Add more documentation

mod_atom_feed

Support for rendering feeds of Atom XML resources.

Todo

Add more documentation

mod_authentication

This module contains the main Zotonic authentication mechanism. It contains the logon and logoff controllers, and implements the various hooks as described in the *Access control* (page 37) manual.

Todo

Add more documentation

mod_backup

mod_backup serves two different purposes: it makes a nightly backup of your files and database, and can also backup/restore individual *resource* items.

Daily backup of database and files

Losing data is bad for business. This applies to your customers as well if you are building sites for them. It is critical to keep backups of any Zotonic sites you develop.

After enabling mod_backup, it will make a backup of the site's data every night at 3 AM. It keeps the last 10 copies of the data, so you have alway a backup to roll back to.

The backups are stored under `user/sites/yoursite/files/backup/`.

The site's media files are stored as a `.tar.gz` file, while the database is stored as an uncrompressed `.sql` file.

We advise to add a cron script to the server so the backup data is copied off of it on a regular interval.

Per-resource backup/restore

Todo

Not yet documented.

mod_base

mod_base is the base module, which acts as a container module holding most of Zotonic basic dispatch rules, *Actions* (page 320) and *Custom Tags* (page 457).

Note that the amount of *templates* has been kept to a minimum in this module, so that sites are free to implement whatever templates they want. For a standard site with commonly used templates, see the *mod_base_site* (page 287) module.

Todo

Add more documentation

See also:

dispatch rules, *mod_base_site* (page 287).

mod_base_site

Implements the templates for a basic Zotonic site, built upon *mod_bootstrap* (page 287) framework.

This module contains templates for different device classes, so that the site is also usable on simpler *text* or *phone* devices.

Todo

Add more documentation

See also:

dispatch rules, *mod_base* (page 287).

mod_bootstrap

Adds support for the Twitter Bootstrap CSS / JavaScript framework.

Todo

Add more documentation

mod_comment

Implements a basic commenting system, enabling commenting on *resources*.

The module has an admin comment overview, allowing the administrator to review the comments or delete them.

To enable commenting in your site, include `_comments.tpl` on your resource's page template, passing an *id* parameter for the resource on which you want users to be able to comment.

mod_component

This is an experimental module. It will be the basis of a new method of loading *components* in HTML pages. Components consist of HTML, javascript, and css. These parts can be dynamically loaded. Layout can/will be done using *mithril.js*.

This module will undergo significant changes, so use at your own risk. It is included to support some parties that use this in production.

mod_contact

Implements a basic contact form, which gets emailed to the configuration value `mod_contact.email`, when submitted.

Todo

Add more documentation

---

mod_content_groups

Todo

Not yet documented.

---

mod_custom_redirect

Enables redirects from unknown hosts and paths to other locations. The other location can be a known path or another web site.

Redirect unknown domains

If the site dispatcher encounters an unknown host name then it notifies all modules with the `#dispatch_host` notification.

The Custom Redirect module observes this notification and checks against a configurable list of domains and redirects. If a domain is matched then the site dispatcher will redirect the user agent to the new location.

The list of domains and their redirects can be configured in the admin: Modules -> Domains and redirects.

The domain must be like what is typed in the browser. Examples of domains are `www.example.org` and `mypc.local:8000`.

Domain names are case insensitive, that is `WWW.EXAMPLE.COM` and `WwW.Example.coM` will both be matched with `www.example.com`. Contrary to the domain name, the path is case sensitive. That is `/ABOUT` and `/About` are two different paths and will need their own redirect rules!

The redirect location can be a complete URL (for example `http://www.example.com/foo/bar.html`) or a path (for example `/about`).

Redirect unknown paths

After the site has been selected, the dispatcher matches the path to the dispatch rules.

When no dispatch rule matches, then the `#dispatch` notification is sent. The *mod_base* (page 287) module observes that notification to check the path against the *page_path* properties of all resources. If *mod_base* (page 287) didn't find match then *mod_custom_redirect* (page 288) will check all custom redirect with an empty domain and a matching path. The visitor will be redirected to the corresponding redirect location.

Permanent or temporary redirects

A redirection can be permanent or temporary. A permanent redirect will be remembered by the visiting browser (and search engines), replacing any occurence of the redirected location. A temporary redirect will not be remembered and be retried on every visit.

See also:

Dispatch rules (page 12), *mod_base* (page 287)

mod_development

Presents various tools for development.

Admin page

After the development module is enabled a menu item *Development* is added to the *System* menu in the admin.

On the development page it is possible to set debugging options, trace template compilation, and test dispatch rules.

Options

This can toggle various development options:

Show paths to included template files in generated templates Checking this will add comments in the compiled templates. The comments will list the exact file included at that point.

Show defined blocks in generated templates Checking this will add comments in the compiled templates. The comments will show the start and end of any template `{% block %} ... {% endblock %}`.

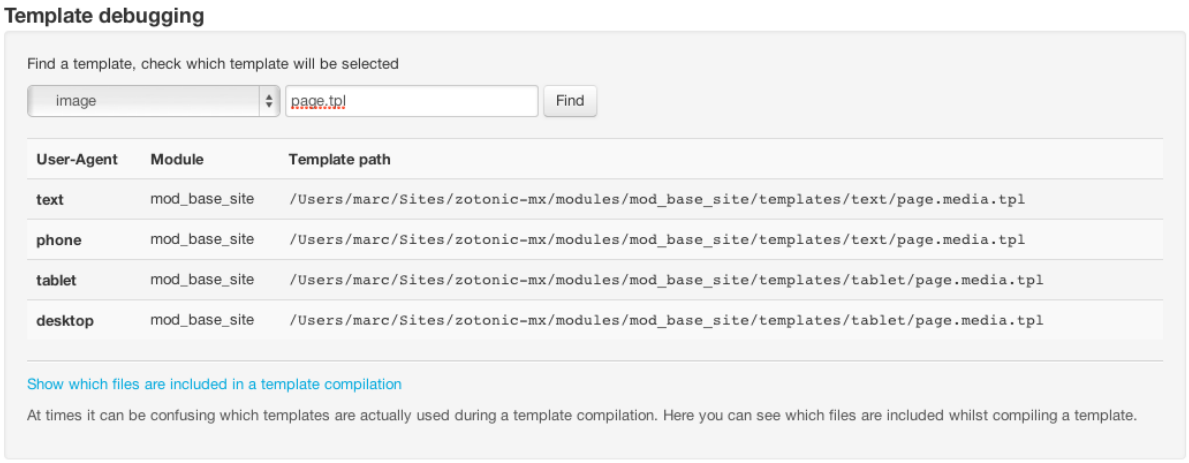
Download css and javascript files as separate files (ie. don't combine them in one url). Checking this will generate separate `<link/>` and `<script/>` tags for all files mentioned in a single `{% lib %}` tag. This makes debugging those files easier but makes loading pages slower as more requests will be done per page.

Enable API to recompile & build Zotonic The api on `/api/development/recompile` can be accessed to trigger a full compilation and cache flush of Zotonic. This checkbox must be checked to enable this api.

Template debugging

The template selection mechanism is quite complicated. It takes into account all modules, their priority, the user-agent class (desktop, tablet, phone or text) and optionally the category of a resource.

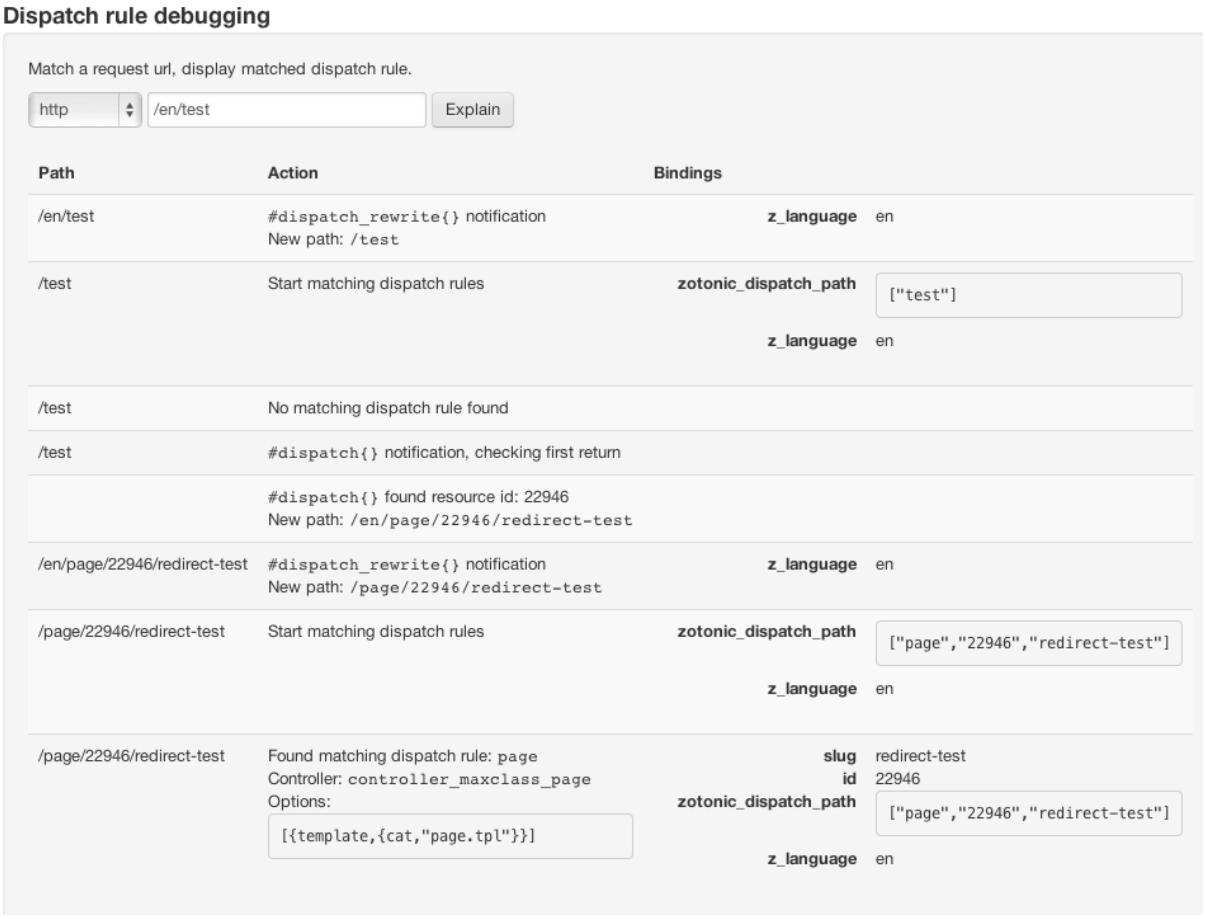
With this debugging tool you can optionally select a category, and fill in the name of the template. Per user-agent class the selected template will be shown.



The second debug option is a page with a live display of all templates being compiled. With this it is possible to get greater insight in the template selection and compilation.

Dispatch rule debugging

With this it is possible to see for a request path which dispatch rules are matched and/or how it is rewritten.



Automatic recompilation

The core Zotonic system starts either `inotify-tools` or `fswatch`, depending on which one is available. You have to install one of these to enable auto-compile and auto-load of changed files.

Note: The system can only scan for changed files if either `inotify-tools` or `fswatch` is installed.

See below for platform-specific installation instructions.

If a changed file is detected then Zotonic will:

- If an *.erl* file changes then the file is recompiled.
- If a *.scss* or *.sass* file changes then `sass` is called to compile it to its *.css* equivalent.
- If a *.less* file changes then `lessc` is called to compile it to its *.css* equivalent.
- If a *.coffee* file changes then `coffee` is called to compile it to its *.js* equivalent.
- If a lib file changes then the module indexer will be called so that any removed or added templates will be handled correctly.
- If a template file changes then the module indexer will be called so that any removed or added template will be handled correctly.
- If a dispatch file changes then all dispatch rules are reloaded.
- If a beam file changes then the module will be loaded. If the beam file is a Zotonic module then it will be automatically restarted if either the function exports or the `mod_schema` changed.
- If the .yrl definition of the template parser changes, then the .erl version of the parser is regenerated. (This will trigger a compile, which triggers a beam load).

Linux installation

On Linux this feature depends on the *inotifywait* tool, which is part of the `inotify-tools` package. For displaying notifications, it uses `notify-send`:

```
sudo apt-get install inotify-tools libnotify-bin
```

Mac OS X installation

On Mac OS X (version 10.8 and higher), we use the external programs `fswatch` and `terminal-notifier`:

```
sudo brew install fswatch
sudo brew install terminal-notifier
```

Configuration options

`mod_development.libsep` Boolean value. If true, *lib* (page 453) files will be included separately instead of in one big concatenated file.

mod_editor_tinymce

Todo

Not yet documented.

mod_email_dkim

Signs outgoing e-mails with DomainKeys Identified Mail Signatures (RFC 6376).

DKIM (DomainKeys Identified Mail) is an important authentication mechanism to help protect both email receivers and email senders from forged and phishing email.

How does it work?

DKIM works by signing each e-mail that Zotonic sends with a private key. The public key part is exposed through a DNS TXT record, with which email receiver can check whether the email actually originated from the domain that it claimed to come from.

This RSA keypair is generated automatically when the module is installed, and the private/public keys are put in the directory `sitename/dkim/`. When the module is active and the keypair has been generated, all outgoing e-mail will be signed.

Note: The generating of the keypair depends on the `openssl` utility to be available in `$PATH`.

DNS configuration

The receiving e-mail server checks the validity of the signature by doing a DNS lookup. To configure DKIM, you will need to add this DNS entry to your domain where you send the mail from.

In the admin, the page `/admin/email/dkim`, available under ("Modules" / "DKIM e-mail setup") provides information how to configure this DNS entry, including the text to copy-paste into the DNS record.

DKIM selector

By default, the DKIM selector is set to the string `zotonic`. This will result in DNS lookups to the `zotonic._domainkey.yoursite.com` domain. You can change the selector name by adding a config value called `site.dkim_selector`.

mod_email_receive

Enables the Zotonic site to receive emails for the site's users. The user's email address is *username@hostname*, where the hostname is the hostname as configured in the *site's config file* (page 11).

Any email that has no valid recipient is rejected.

See also:

mod_email_relay (page 292), *E-mail handling* (page 56).

Todo

Add more documentation

---

mod_email_relay

Enables the Zotonic site to *relay* emails for the site's users to their real email addresses.

The user's email address is *username@hostname*, where the hostname is the hostname as configured in the *site's config file* (page 11). Any mails to those addresses get forwarded to the user's email address, as configured in the user *resource*.

Any email that has no valid recipient is rejected.

See also:

mod_email_receive (page 292), *E-mail handling* (page 56).

Todo

Add more documentation

---

mod_email_status

This module tracks for all outgoing email addresses:

- If emails are successfully sent
- Number of emails sent
- Number of errors
- Number of bounces
- Latest error message

With this it will be much easier to get feedback on all outgoing email.

mod_export

Provides a generic framework to export *resources*.

Currently, when installed, the module exposes a url `/export/csv/:id` which exports the resource with the given ID to a CSV (Comma Separated Value) format. Only the following resource fields are currently exported: title, summary, created, modified, page_url_abs.

Currently, CSV is the only supported export format, but the module is set up in such a way that other export formats are easy to add.

mod_facebook

Plugs into the authentication / signup system to enable Facebook login.

Module is still largely undocumented, but this is what was said on the mailing list about it:

The url of your site needs to be correctly configured in the Facebook app configuration at Facebook for your site to work.

The current (hard coded) test site has the url http://127.0.0.1:8000/ configured. Use the config keys:

- `mod_facebook.appid`
- `mod_facebook.appsecret`

Both values you can find when you register your site (as an application) at Facebook.

If you need extended permission for the acquired token, you can use the `mod_facebook.scope` configuration key to set the scope. Note that the module always will need the "email" permission for the login to work.

Todo

Unfinished doc

mod_filestore

Support for storing uploaded and generated images and documents on external services.

Overview

This module stores uploaded files and generated preview-images on an external S3-compatible service. It listens for medium and file related notifications for any newly uploaded or generated files.

If a file is added then the file is queued in an upload queue. After a delay a separate process polls this queue and will upload the file to the external service.

If a file is needed and not locally available then the mod_filestore module will check its file registry to see if the file is stored on an external service. If so then then a *filezcache* process is added and a download of the file is started.

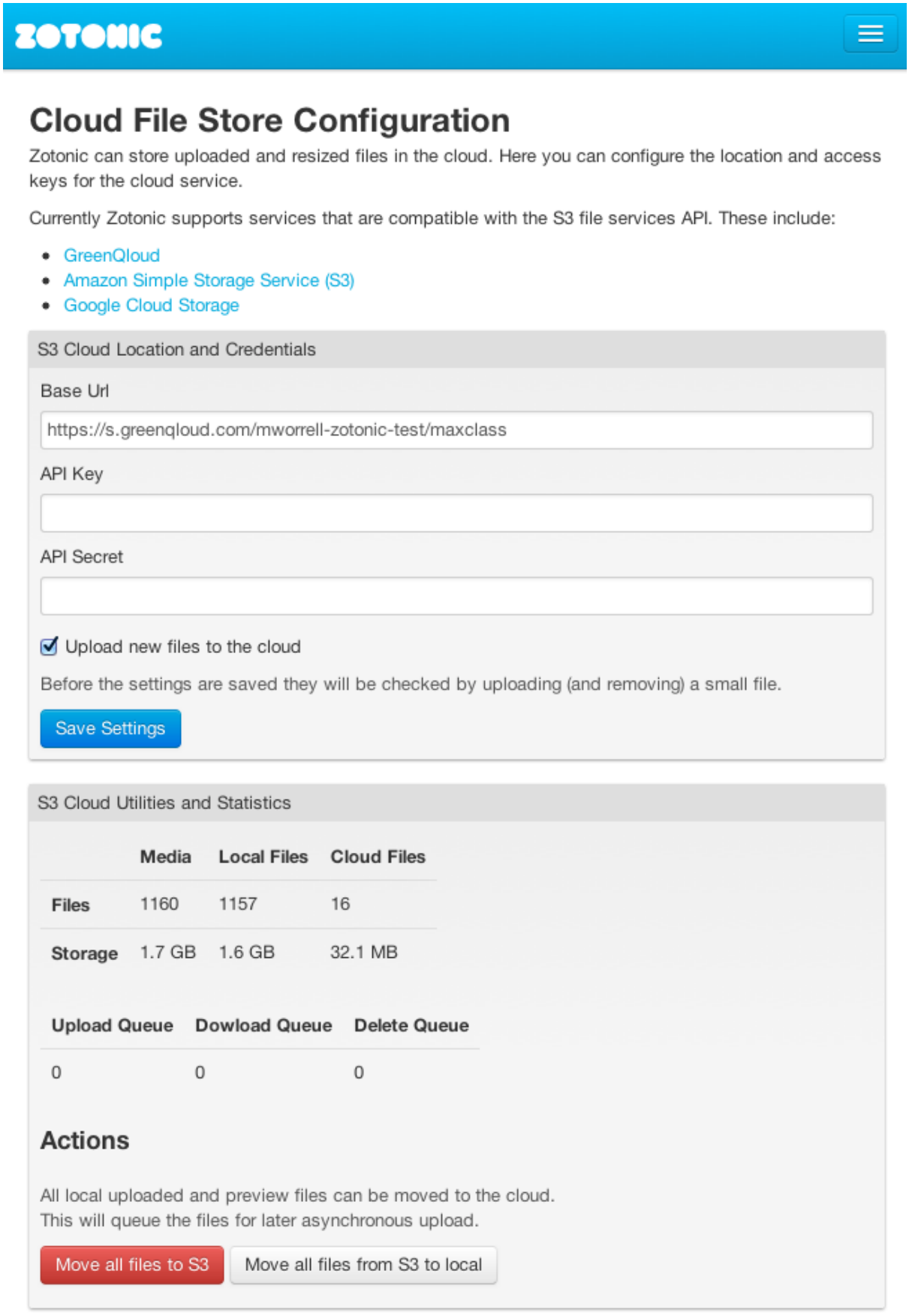
The file is served from the filezcache whilst it is being downloaded.

The *filezcache* will stop the entry after a random amount of time—if the entry was not recently used.

Configuration

After the *mod_filestore* is enabled an extra menu entry *Cloud File Store* is added to the *System* menu in the admin.

Selecting the menu will show the configuration panel for the Could File Store.



Here you can define where the files should be stored and give the credentials to access the storage.

If you save the url and credentials then the system will try to upload a small file to the remote storage. If it succeeds then the configuration is saved. If it does not succeed then an error message will be displayed and the configuration will not be changed.

It is possible to (temporarily) disable uploading new files by unchecking the checkbox *Upload new files to the cloud*.

File deletion

You can also configure file deletion behaviour, i.e. what should happen when a file is removed from Zotonic. You can choose to immediately remove the file from the filestore, not delete it at all (to make your store immutable) or delete the file after a certain delay (to be able to restore accidentally deleted files).

Statistics

The system shows statistics:

Media All *medium* records and a sum of the sizes. A single medium record can have 0, 1 or 2 files attached.

Local Files These are all files found in the *files* directory, this includes files that won't ever be uploaded.

Cloud Files All files registered to be on any cloud service. This is extracted from the database and not by scanning the remote cloud service.

Queues These are the queues being processed by *mod_filestore*. On a quiet (stable) system they are usually empty.

Moving files

It is possible to move (almost) all files from the local file system to the cloud. And vice versa, from the cloud to the local file system. This is useful when starting or changing the cloud storage location.

If a file is moved to the cloud then it is first placed in the filezcache. The filezcache will start purging the files if the cache is bigger than configurated in the filezcache application (default 10GB for all sites combined).

The system waits 10 minutes before a queued file is uploaded. This period is meant for a *cool down* of the file, as in the first moments after an upload some resize and preview operations will take place. The delay makes it less probable that a freshly uploaded file vanishes (to the cache) whilst a preview-generation is starting.

Notifications

The *mod_filestore* hooks into the following notifications, whose definitions can be found in `zotonic_file.hrl`:

`#filestore{}` Hooks into the Zotonic file management notifications to upload, delete or lookup files. This will trigger downloads of external files and interfaces to the filezcache.

`#filestore_credentials_lookup{}` Maps a local path and optional resource id to a service, external location and key/password for that external service. This can be used to store different resources on different external services.

`#filestore_credentials_revlookup{}` Maps a cloud file service and location to a key, password and request location.

`#medium_update_done{}` Queues newly inserted medium files into the upload queue.

`#admin_menu{}` To add the *Cloud File Store* menu to the admin.

Applications

The filestore uses the *s3filez* and *filezcache* Erlang applications.

s3filez

This application is used for uploading, downloading and deleting files on S3 compatible services. It provides asynchronous services and is compatible with the filezcache application. It is also able to stream files to and from the external S3 service, this makes it possible to have start serving a file before it is downloaded to the filezcache.

filezcache

This application manages a cache of downloaded files. The cache is shared between all sites. Every cache entry is managed by its own process, which can stream newly received data directly to any requesting processes.

The filezcache keeps a presistent *disk_log* with a description of all files in the cache. This log is read on startup to repopulate the cache with already present files. For each file the size and a hash is stored to check cache consistency.

The filezcache has a garbage collector. It keeps a pool of randomly selected cache entries, from which it will elect randomly processes to be garbage-collected. The processes themselves will decide if they will stop or not.

After a cache process stops it will keep running for a short period to handle late incoming requests.

Filezcache entries are started by the *mod_filestore* and filled by either moving a local file to the cache or by s3filez download processes.

Todo

The statistics are generated dynamically, which is not a good idea with many files. This will be changed.

See also:

m_filestore (page 419).

mod_import_csv

Module which adds "import CSV" button to the admin status screen.

The dropbox folder of the site is also watched for CSV files.

To determine whether it can import a file, it uses a notification:

```
#import_csv_definition{basename, filename}
```

If the notification is not returning anything, it tries to map the columns found in the first row of the CSV file to *resource* column names.

Todo

Add more documentation

---

mod_import_wordpress

Import wordpress .wxr files in your site.

Wordpress, the famous PHP blog system, is able to export all of its contents into a `.wxr` XML file. With this module, you can import such a .wxr file in Zotonic, when you want to migrate your blog. The import routine will import all posts, keywords, categories, authors and images.

Once you enabled the module, the import button lives in the admin in the "status" tab.

The import module works incrementally: if you import a .wxr file twice, it will update the changed entries and add new entries. If you delete posts, a next import will not re-create the entries: it will remember that you have deleted them. You can override this behaviour by checking the "import previously deleted content" button.

Note: YMMV with importing 2 .wxr files from different blogs: as the unique keys of the entries are based on the wordpress numeric ids, it is possible that content from the second one will overwrite content from the previous import.

mod_instagram

Todo

Not yet documented.

mod_l10n

Localization of Zotonic. Provides lookups for country, month, week names.

Todo

Add more documentation

mod_linkedin

Todo

Not yet documented.

mod_logging

Adds log views to the admin.

Log messages

Any message from Zotonic which uses the `zInfo` or `zWarning` macros is logged in the log view, in real time.

E-mail log

The e-mail log is a separate view, which lists which email messages have been sent to which recipients. Any mail that gets sent gets logged here.

Todo

Add more documentation

mod_mailinglist

This module implements a mailing list system. You can make as many mailing lists as you like and send any page to any mailing list, including confirm mail and unsubscribe page.

Mailing lists are pages of the category *mailinglist*, which is installed by the module upon activation. In the admin there is support for managing these mailing lists where you can import, export, add or delete recipients.

For details on configuring e-mail sending and receiving in Zotonic, see *E-mail handling* (page 56).

Including the subscribe custom tag on your pages

The module includes the signup tag tag (and template) that you can use on your site.

When you want to add a subscribe template to your page then you will need the following scomp include in your template:

```
{% mailinglist_subscribe id=mailing_id %}
```

Where `mailing_id` should be set to the page id of your mailing list. This scomp includes the template `_scomp_mailinglist_subscribe.tpl`. The subscription form itself can be found in the template "_mailinglist_subscribe_form.tpl". You should overrule the latter template when you want to add or remove fields from the subscription form.

Pages for the mailing list category

The mailinglist module predefines the following dispatch rules for the mailinglist:

```
/mailinglist/1234
/mailinglist/1234/my-mailinglist-slug
```

Where 1234 stands for the id of your mailinglist (which is obviously another id).

The mailinglist page is very simple. It shows the title, summary and body of the mailinglist, followed by a subscribe form and finally a list of other mailing lists.

Template used for the e-mails

All e-mails use the template-mailing_page template. It is a very simple template that just tells why the recipient received the e-mail, refers to the sent content page on the Internet and finally shows the title, the summary and the body of the sent page.

In the footer there is a link to the unsubscribe page.

All e-mail templates extend from the template-email_base template. The following templates are used for e-mails:

Template	Description
template-email_mailinglist_confirm	Sent after subscribing to a mailing list, requests to click on an url to confirm the subscription.
template-email_mailinglist_goodbye	Sent after unsubscribing from a mailing list.
template-email_mailinglist_welcome	Sent after subscribing and confirming the subscription.

Sending mailings in the admin

On the resource edit page of any page (remember: the mailinglist module can send *any* resource as a mailing!) there is an link in the right column called *Go to the mailing page*.

The mailinglist recipients page

Each mailinglist has a special page in the admin that lets you view the recipients that are part of the list. On that page you can perform varions functions on the recipients of the list.

- *Add new recipient* - Opens a dialog to add a single recipient to the list.
- *Download recipient list* - Downloads a `.txt` file with all the e-mail addresses and name details of all recipients.
- *Upload recipient list* - Upload a new file with recipients. Each e-mail address goes on its own line. There is a checkbox which lets you clear the list before the import, effectively overwriting all recipients in the list.
- *Clear recipient list* - After a confirmation, this removes all recipients from the list.
- *Combine two lists* - this opens a dialog which lets you combine two lists. Using this new dialog, the recipients of two lists can be combined according to the three set operations union, subtract and intersect.

The mailing status page

From the mailing status page you can send the current resource to a mailing list, to the test mailing list or to an email address.

Sending mailings

The status page lists every mailing list in the system. On each row you see how many recipients the list has, and the status, e.g. if the mailing has already been sent to this list or not.

Todo

finish description of the mailing status page

Mailings are only send when the to be send page is published and publicly visible. The page should also be within its publication period.

You can schedule a mailing by publishing the page but setting its publication start date to the date and time you want your mailing to be send. The mailing list module checks every hour if there are any mailings ready for sending.

An exception is made for the test mailing list, mailings to that mailing list are always sent.

mod_media_exif

When uploading a file, this module extracts properties from the uploaded media and sets them in the resource.

Properties extracted from the uploaded file are:

- GPS location
- Crop center (derived from the focusing area)
- Orientation
- Date (start, end and original publication date)

mod_menu

Create nested navigation menus for your site.

Activating the module in the admin enables a "menu" item in the admin navigation under "content", which lets you define a simple menu. Every item in the menu references a Zotonic page and can be looked up using the autocompletion widget.

This menu can be rendered in the frontend with the scomp-menu custom tag.

It will use the _menu.tpl template which is by default able to render a Twitter Bootstrap compatible menu structure using nested `<ul>` elements.

To implement a different navigation menu, override the `_menu.tpl` in your project and create new markup.

Domain model

The *domain model* for this module is the following:

The module creates a new category named *menu*. This allows one to create multiple menus in a single site. Its edit page in the admin contains the hierarchical menu editor.

The menu resource that is accessible from the admin page (*Content* > *Menu*) is the resource with the unique name `main_menu`.

See also:

The filters *menu_flat* (page 392), *menu_subtree* (page 392) and *menu_trail* (page 393).

mod_mqtt

Enables MQTT support in a site.

MQTT is a machine-to-machine (M2M)/"Internet of Things" connectivity protocol. It was designed as an extremely lightweight publish/subscribe messaging transport. It is useful for connections with remote locations where a small code footprint is required and/or network bandwidth is at a premium. For example, it has been used in sensors communicating to a broker via satellite link, over occasional dial-up connections with healthcare providers, and in a range of home automation and small device scenarios (source and more information: MQTT.org)

MQTT uses a simple message broker to route messages from publishers to multiple subscribers.

A quick overview of MQTT

Publish/subscribe

With MQTT messages are published to topics. All subscribers to a topic will then receive the message. A topic is a string, much like a file-path, for example: `truck/0001/temperature`

A subscriber can directly subscribe to a topic, or can use wildcards to subscribe to related topics. For this the wildcards + and # can be used. + matches exactly one word between the slashes, and # can be used at the end of a pattern to match all sub-topics.

Examples of subscription patterns:

- `truck/0001/temperature` matches the temperature publications of truck 0001.
- `truck/+/temperature` matches all temperature publications for all trucks.
- `+/+/temperature` matches all temperature publications for all trucks and other things with a temperature
- `truck/0001/#` matches all publishes to `truck/0001` and all its sub-topics

Retained messages

A publisher can publish a *retained* message to a topic. When publishing all current topic subscribers will receive the message. Above that, if a new subscription is made to the topic, then all retained messages are sent to the new subscriber.

Quality of service

MQTT has three levels for the quality of message delivery. These are used when sending messages between machines. The levels are:

- Level 0: send the message, no reception acknowledgments are reported.
- Level 1: on receipt a single ack is sent back to the publisher
- Level 2: a double handshake is performed

For most communication level 0 is used.

Wills

A client can set a *last will* message and topic. This is a message that will be published to the topic at the moment the client is unexpectedly disconnected.

MQTT in Zotonic

Zotonic has a central MQTT message broker. Optionally clients can connect to this broker using the normal MQTT protocol.

The broker is used for internal publish/subscribe support.

Each open HTML page can also have a local (simplified) broker. The system can relay messages between the brokers on open pages and the central broker in Zotonic. In this way it is possible for HTML pages to have their own local publish/subscribe system and also subscribe or publish to topics on the central broker.

As the central broker is shared between sites it is even possible to publish/subscribe between different sites. In the future it will be possible to bridge the brokers between servers.

Predefined topics

Currently the following topics are defined:

Topic	Description
public	Freely accessible topic, both for subscribe and publish
test	Test topic. If you publish here then mod_mqtt will log debug message.
user	Topic available for any authenticated user
site/sitename	Root for a site
site/sitename/public	Freely accessible within the site
site/sitename/test	Test topic, freely accessible within the site
site/sitename/user	Topic available for any authenticated user of the site *sitename*
site/sitename/user/UserId	Topic available for a specific user of the site *sitename*
site/sitename/session	HTML pages can subscribe, admins can publish
site/sitename/session/SessionId	Topic to relay information to a specific session
site/sitename/pagesession	HTML pages can subscribe, admins can publish
site/sitename/pagesession/PageId	Topic to relay information to a specific page (in a browser)

Topics and namespaces

To make it easier to write generic software, without changing topic names, some namespace conventions and mappings are introduced.

The following topics are expanded

Topic	Expansion | Description	
~site	site/mysite	The context's site root topic
~session	site/mysite/session/oLfVVaT299zpSjlGb5Im	The topic for the current session
~pageses-sion	site/mysite/pagesession/vWCUKL9QKmfLxotWoFZx	The topic for the current HTML page in the browser
~user	site/user/1234	The topic for the current user

Note that there are not automatic subscriptions for session, pagesession and user topics. All subscriptions need to be added explicitly.

Access control

All topics have access control added. For this an extra ACL object `#acl_mqtt{}` defined, with the actions `publish` and `subscribe`. Modules can observe the usual `#acl_is_allowed{}` notification to add access control to topics.

Subscribing modules

Modules can automatically subscribe to topics. This is done by adding specially named functions.

For example, the following function subscribes to the topic `site/sitename/test`:

```
-export([
    'mqtt:~site/test'/3
]).

'mqtt:~site/test'(Message, ModulePid, Context) ->
    lager:debug("mqtt:~site/test received: ~p", [{Message, ModulePid, z_
↪context:site(Context)}]),
    ok.
```

Here *Message* is the received `#mqtt_msg{}`, and *ModulePid* is the process id of the running module.

The function will be called from within a process that is subscribed to the topic.

Erlang API

Subscribe a function F in a module M to a topic:

```
-spec subscribe(binary()|string(), {atom,atom}, #context{}) -> ok | {error, eacces}
↪.
z_mqtt:subscribe(Topic, {M,F}, Context)
```

This starts a process that will subscribe to the topic and call the function whenever a message is received. The topic can have wildcards, though access control applies and the result `{error, eacces}` will be returned if access is denied, `ok` will be returned on a succesful subscription.

Subscribe the current process to a topic:

```
-spec subscribe(binary()|string(), #context{}) -> ok | {error, eacces}.
z_mqtt:subscribe(Topic, Context)
```

When the process stops it will automatically be unsubscribed. The process will receive messages `{route, #mqtt_msg{}}`.

Subscribe another process to a topic:

```
-spec subscribe(binary()|string(), pid(), #context{}) -> ok | {error, eacces}.
z_mqtt:subscribe(Topic, Pid, Context)
```

To unsubscribe, use `z_mqtt:unsubscribe` with the same arguments as during subscription.

To publish a message:

```
-spec publish(binary()|string(), term(), #context{}) -> ok | {error, eacces}.
z_mqtt:publish(Topic, SomeData, Context)
```

JavaScript API

The JavaScript API uses callback functions:

```
pubzub.subscribe("~pagesession/foo/#", function(topic, msg) { console.log(topic,␣
↪msg); });
pubzub.publish("~pagesession/foo/bar", "hello world");
```

If the received message was relayed from the server then it is an object:

```
{
    ubf_type: ubf.TUPLE,
    _record: "z_mqtt_payload",
    version: 1,
    site: "yoursitename",
    user_id: 23,                    // The id of the user sending the message
    encoding: "ubf",                // The way the payload was encoded
    payload: ...                    // The decoded payload
}
```

The transport between the server and the browser always uses UBF(A). Most decoded values will be an object with an extra `ubf_type`; always use the method `.valueOf()` to get the primitive type of the object.

You will need to include the following JavaScript files:

```
{% lib
        "js/qlobber.min.js"
        "js/pubzub.js"
%}
```

The file `js/modules/ubf.js` should already have been included, as it is used by `zotonic-1.0.js`.

Connection will

Currently a simple version of the *lastwill* is available for JavaScript. This sets a topic and message to be sent when the page process stops.

Multiple wills can be set. Currently it is not possible to remove a will, though that will change in the near future.

Example:

```
var will_id = pubzub.lastwill("~site/goodbye", "thanks for the fish");
```

Quality of service

Currently there is no quality of service implemented for the JavaScript API and relay. The server side page process will buffer all messages till the browser connects to the page session. This happens on connects with comet, WebSocket, and postbacks.

On the browser all messages are queued and sent one by one to the server. This uses either the WebSocket connection or the postback interface.

Enabling the MQTT listener

MQTT can listen on a port for incoming connections. Per default the listener is disabled.

Configuration

The MQTT listener is configured in the `priv/erlang.config`. If this file is missing then it can be copied from `priv/erlang.config.in`.

The following section defines the Zotonic authentication module, access control, and a listener on the standard MQTT port 1883:

```
{emqtt, [
   {auth, {z_mqtt_auth, []}},
   {access_control, {zotonic, []}},
   {listeners, [
       {1883,  [
           binary,
           {packet,        raw},
           {reuseaddr,     true},
           {backlog,       128},
           {nodelay,       true}
       ]}
   ]}
]},
```

Authentication

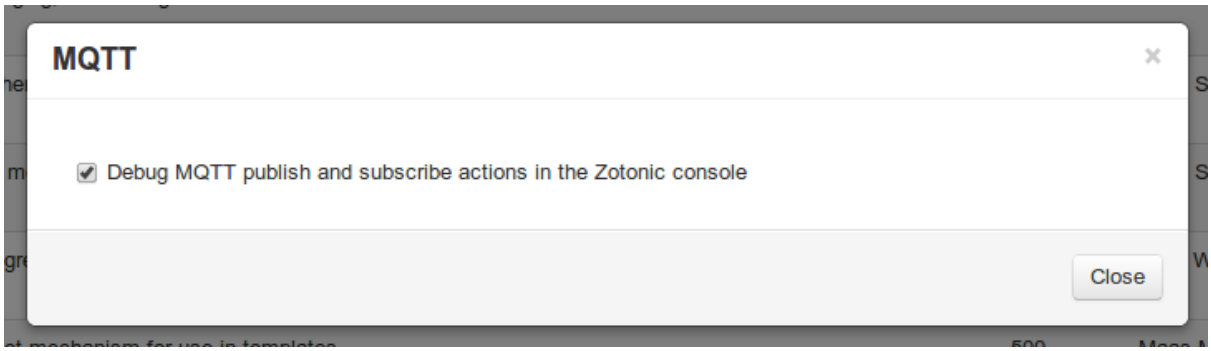
All connections must authenticat themselves using an username and password. The username is postfixed with the hostname of the user's site, for example: `jantje@foobar.com`. In this way Zotonic knows which site the user belongs to.

If no matching site can be found, or if no hostname is given, then Zotonic will try to authenticate against the default site.

Debugging

To see which topics are being subscribed and published to, you can configure `mod_mqtt` to print debug messages on the Zotonic console whenever a publish or subscribe occurs.

To do so, go to the modules overview in the admin interface, and scroll down to `mod_mqtt`. Then, click on the "configure" button in the row in the right and tick the checkbox to enable these debug messages.



mod_oauth

Module providing OAuth authentication support for the *API services* (page 51) of Zotonic.

OAuth allows resource owners (admins, developers) to authorize third-party access to their content or modules without sharing their credentials.

If you need to provide access to one of your own modules, your first step will be to create a service (sub)module. See *API services* (page 51) for details.

How the OAuth module operates

This module hooks in the `service_authorize` callback to check requests handled by *controller_api* (page 349) for OAuth authentication tokens.

Requests processed by this module are checked for the `Authorization: OAuth` request header and if it is found, it checks for a valid token and verifies the request signature. When this is all done, it looks which OAuth application (the *consumer key*) is being used and checks if the requested API service is in the list of allowed services for that particular application.

Defining OAuth applications

The module adds an admin menu for defining OAuth applications (*consumers*). For each consumer, you can specifiy which OAuth calls are permitted.

See also:

controller_api (page 349)

Todo

Add more documentation

mod_oembed

Makes media *resources* from embeddable URLs through the OEmbed protocol.

When activated, in the "create media / page" dialog, an extra tab is added which allows you to paste a media URL from services like YouTube, Vimeo or any other service which supports OEmbed.

A saved media resource has a thumbnail image which is downloaded from the OEmbed service and embedded in the resource. Furthermore, the resource's *medium* record has an `oembed` field which contains the full JSON response of the request. The `oembed` field looks like this:

```
"oembed": {
  "type": "video",
  "version": "1.0",
  "provider_name": "Vimeo",
  "provider_url": "http://vimeo.com/",
  "title": "Heli Filming Showreel",
  "author_name": "Hot Knees Media",
  "author_url": "http://vimeo.com/hotknees",
  "is_plus": "1",
  "html": "<iframe src=\"http://player.vimeo.com/video/20898411\" width=\"640\"␣
↪height=\"362\" ...",
  "width": 640,
  "height": 362,
  "duration": 106,
  "description": "description..",
  "thumbnail_url": "http://b.vimeocdn.com/ts/138/106/138106290_640.jpg",
  "thumbnail_width": 640,
  "thumbnail_height": 362,
  "video_id": 20898411
}
```

So, to display the HTML of an OEmbedded medium, you would do the following in a template:

```
{{ id.medium.html }}
```

The module also supports the use of the *media* (page 453) tag:

```
{% media m.rsc[id].o.depiction.medium %}
```

Note however, that setting dimensions on this media tag is not supported for OEmbed, as the embed width/height is always taken from the provider.

Configuration options

The following *m_config* (page 415) options are supported:

`oembed.maxwidth`

> Requests the OEmbed service to return an HTML embed code with the requested maximum width. Defaults to 640.

`oembed.maxheight`

> Requests the OEmbed service to return an HTML embed code with the requested maximum height. Not set by default.

Todo

Extend documentation

mod_rest

Makes models accessible using a RESTful interface.

Currently there are endpoints implemented for *resources* only, at the following URLs:

```
/rest/rsc/:id
/rest/rsc/:format:id
```

format can be one of *json* for JSON encoding or *bert* for BERT.

The controller accepts GET, PUT and DELETE requests.

- GET returns the dump of the *resource*
- PUT updates the resource with the information provided
- DELETE removes the resource.

All of these methods use the standard Zotonic *authentication* (page 37) checks.

See also:

controller_rest_rsc (page 360)

Todo

Add more documentation

mod_search

mod_search implements various ways of searching through the main resource table using *m_search* (page 432).

Configuration

There are two *site configuration variables* (page 491) to tweak PostgreSQL text search settings.

mod_search.rank_behaviour

An integer representation to influence PostgreSQL search behaviour.

Default: `37` (`1 | 4 | 32`)

mod_search.rank_weight

A set of four numbers to override relative weights for the ABCD categories.

Default: `{0.05, 0.25, 0.5, 1.0}`

The following searches are implemented in mod_search:

Name	Description	Required arguments
featured	List of pages, featured ones first.	
featured	List of pages in a category, featured ones first.	cat
featured	List of pages in a category having a certain object, featured pages first.	cat, object, predicate
latest	The newest pages.	
latest	The newest pages within in a category.	cat
upcoming	Selects pages with future date_end.	
finished	Selects pages with a past date_end.	
ongoing	Pages with past date_start and future date_end.	
autocomplete	Full text search where the last word gets a wildcard.	text
autocomplete	Full text search where the last word gets a wildcard, filtered by category.	cat, text
fulltext	Full text search. Returns `{id,score}` tuples.	text
fulltext	Full text search, filtered by category. Returns `{id,score}` tuples.	cat, text
referrers	All subjects of a page.	id
media_category_image	All pages with a medium and within a certain category. Used to find category images.	cat
media_category_depiction	All pages with a depiction edge to an image. Used to find category images.	cat
media	All pages with a medium, ordered by descending creation date.	
all_bytitle	Return all `{id,title}` pairs for a category, sorted on title.	cat
all_bytitle_featured	Return all `{id,title}` pairs for a category, sorted on title, featured pages first	cat
all_bytitle	Return all `{id,title}` pairs for a category without subcategories, sorted on title.	cat_is
all_bytitle_featured	Return all `{id,title}` pairs for a category without subcategories, sorted on title, featured pages first.	cat_is
match_objects	Returns a list of pages with similar object ids to the objects of the given resource with the given id. Returns `{id, rank}`. Accepts optional cat parameters for filtering on category.	id
match_objects_cats	Returns a list of pages with similar object ids or categories. Returns `{id, rank}` tuples. Accepts an optional `cat` parameter for filtering on category.	id
query	Very powerful search with which you can implement almost all of the other search functionality. See: *Search* (page 27)	
archive_year	Returns an overview on publication year basis, for a specified category. Every row returned has parts: “as_date”, “year” and “count”. The order is descending, newest year first.	cat
archive_year_month	Return a grouped “archive” overview of resources within a category. The result is a double list, consisting of `[ {year, [ months ] }]`. The result is grouped on publication year and month, and includes counts. The order is descending, newest year first.	cat
keyword_cloud	Return a list of `{keyword_id, count}` for all resources within a given category. The list is ordered on keyword title. Default predicate is `subject`, default category is `keyword`. Change optional `keywordpred` and `keywordcat` to create a different cloud.	cat, keywordpred, keywordcat
previous	Given an id, return a list of “previous” ids in the given category. This list is ordered by publication date, latest first.	id, cat
next	Given an id, return a list of “next” ids in the given category. This list is ordred by publication date, oldest first.	id, cat

See also:

Search (page 27)

mod_seo

Adds basic search engine optimization to the base templates and provides an admin interface for the SEO modules.

Google Analytics

To enable Google Universal Analytics tracking on your Zotonic website, go to `http://yoursite/admin/seo` in your browser and enter your Google Analytics tracking code.

Zotonic automatically supports the User ID Analytics feature by adding the user ID to the tracking code. You have to enable User ID in your Analytics account first.

Extra parameters

If you wish to add extra Google Analytics parameters, override the `_ga_params.tpl` file and add the parameters:

```
{#
    Override this template to provide extra Google Analytics parameters.
    See https://developers.google.com/analytics/devguides/collection/analyticsjs/
↪field-reference
#}
{
    {% if m.acl.user %}
        "userId": "{{ m.acl.user|escapejs }}",
    {% endif %}
    "sessionControl": "start"
}
```

Todo

Add more documentation

mod_seo_sitemap

Creates a `sitemap.xml` file for your site, containing links to all publicly accessible pages.

mod_seo_sitemap creates a `sitemap.xml` file for which is used by the Google bot to index your site. By default, its generated sitemap.xml lists all pages of the categories text, event, location, collection and person in the system which are publicly viewable.

Creating a sitemap without using mod_seo_sitemap

If you have a site with mostly static URLS, it might be easier to create your own sitemap XML file. If you create a dispatch rule like this:

```
{sitemap_xml, ["sitemap.xml"], controller_template, [{template, "mysitemap.tpl"},
↪{content_type, "text/xml"}]}
```

You will be able to write your own sitemap.xml file (in a mysitemap.tpl template) without using mod_seo_sitemap. This might be handy if you serve mostly static files.

mod_signal

mod_signal allows template developers to create pages with highly interactive behaviour. It allows one to emit so called signals. Signals are asynchronous events with attributes. Other pages can connect to the signature of a signal. When that signal is fired, the specified action is triggered. This makes it possible to create interactive pages without writing a line of erlang code.

Signals

A signal is an erlang tuple which can be emitted either by erlang code:

```
mod_signal:emit({example, [{test, "a"}, {session, 123}]}, Context)
```

or by using the emit action in a template:

```
{% button action={emit signal={example test="a" session=123}} %}
```

Signals themselves don't do much. The fun part is connecting other stuff to signals. When a signal is emitted `mod_signal` examines if there is something which is interested in this particular signal. It could be that there are actions connected to signals of type `example`. When such a signal is emitted the registered action is performed. This can be done with the `connect` action.

For example:

```
{% wire action={connect signal={example session=123} action={growl text="example"}}
↪ %}
```

The action above will trigger the grow action when a signal of type example and with property `{session, 123}` is emitted. It is possible to make multiple matches on the property of a signal.

Actions

The module has three actions which can be used to connect and disconnect to signals, and an action to emit signals.

connect Connect actions to signals. The actions are executed when the signal is emitted. For more information see: action-connect.

emit Emit a signal from a template. See: action-emit.

disconnect Disconnect all actions. The actions will no longer be executed when the signal is emitted. See: action-disconnect.

Model

Model `m.signal` makes it possible to use the data of the emitted signal inside a template. See: *m_signal* (page 433).

See also:

action-connect action-emit action-disconnect *m_signal* (page 433)

mod_signup

This module presents an interface for letting users register themselves.

Notifications

signup_form_fields Fold for determining which signup fields to validate. This is an array of `{Fieldname, Validate}` tuples, defaulting to `[{email, true}, {name_first, true}, {name_surname_prefix, false}, {name_surname, true}]`. Observers can add / remove fields using the accumulator value that is passed into the notification.

identify_verification{user_id=UserId, identity=Ident} Sent verification requests to non verified identities.

signup_check Fold for the signup preflight check. Allows to add extra user properties or abort the signup.

If no `{ok, _Props1, SignupProps}` is returned, but `{error, Reason}`, the signup is aborted.

signup_done{id=Id, is_verified=IsVerified, props=Props, signup_props=SignupProps} Fired when a signup procedure is done and a user has been created.

signup_confirm{id=UserId} Fired when a user has signed up and confirmed his identity (e.g. over email)

signup_confirm_redirect{id=UserId} Decide to which page a user gets redirected to after signup

Config: Disabling confirmation email

Set the configuration value `mod_signup.request_confirm` to `false` to disable the signup confirmation process.

Config: Using the user's e-mail address as username

By setting a configuration value, it is possible to use the entered email address as the username.

Set the configuration value `mod_signup.username_equals_email` to `true`.

This makes the username equal to the email address, so that the user can log in using his email address instead of a separate user name. Note that when you allow a user to change his email, take care to update the `{username_pw, {Username, Password}}` identity as well, otherwise the username remains equal to the old email address.

Config: setting the category for new users

By default, users created through the signup process will become *resources* of the category *person*. This can be changed by setting the configuration value `mod_signup.member_category` to the name of a different category.

Config: setting the content group for new users

By default, users created through the signup process will become *resources* in the content group *default_content_group ‘. This can be changed by setting the configuration value ‘‘mod_signup.content_group‘* to the name of a different content group.

Config: setting the visibility of new users

By default, the `visible_for` property of the new users's *resource* will be set to 0, meaning world-viewable. To control the value of the `visible_for` flag on signup, set the configuration value `mod_signup.member_visible_for` to either 1 (visible for other logged in members), 2 (visible for "group members") or 3 (visible only for the user itself).

Todo

Add more documentation

mod_ssl

The mod_ssl module adds HTTPS support. After enabling mod_ssl the logon window and other secure pages will be served using HTTPS.

SSL support can be switched on for each site separately. Virtual hosting of sites via HTTPS is possible. In order to get working SSL connections you need to enable a module providing SSL certificates. Module *mod_ssl_self_signed* (page 314) is an example of such a module. The role of this module is secure dispatching.

Configuration

There are four configurarion options. They can be set in the admin. All four are optional and will be either set or replaced with a default when not set.

mod_ssl.is_secure When this configuration is set to something else then `0` or `false` then mod_ssl will ensure that sessions started while using HTTPS are secured and only valid on HTTPS.

The *autologon* cookie will also be HTTPS only (if set from a HTTPS connection).

Besides that mod_ssl will ensure that, after using HTTPS, all following pages will be served using HTTPS. Unless specifically specified otherwise in the dispatch rules.

This is done by adding the `{ssl, keep}` option to all dispatch rules that do not have an `ssl` option already.

mod_ssl.is_ssl Force all dispatch rules to use `{ssl, true}`, unless specified otherwise. Use this in combination with `mod_ssl.is_secure` to ensure serving a site over HTTPS.

mod_ssl.is_permanent When set to `true` it makes http to HTTPS protocol redirects for dispatch rules which use `{ssl, true}` to be permanent redirects. The default setting is `false` which will make redirects temporary redirects.

Serving a page via SSL

The *dispatch rule argument* (page 12) `ssl` defines if a page will be served over HTTPS or HTTP.

There are three variations:

{ssl, any} Keep the same protocol as before, don't switch beteen HTTP and HTTPS. This used for lib and image files.

{ssl, true} Force a switch to HTTPS. When accessing the page using http then the page will be reloaded using HTTPS. This is useful for logon, logoff and other authentication or secure pages.

{ssl, false} Force a switch to HTTP. When accessing the page using HTTPS then the page will be reloaded using HTTP. This is useful for pages with embedded video or other non HTTPS content.

When the `ssl` option is not specified then it defaults to `{ssl, false}`. Unless the `mod_ssl.is_secure` option is set, then default is `{ssl, any}`.

Example of a page delivered using HTTPS:

```
{logon,  ["logon"], controller_logon, [{ssl, true}]}
```

And of a dispatch rule that should keep the protocol, in this case the lib controller used for serving css, javascript and other static lib files:

```
{lib, ["lib",'*'], controller_lib, [{ssl, any}]}
```

Notifications

When you want to implement your own certificate handling you have to implement a notification handler which returns the certificates to the underlying HTTPS server. This can be needed when you have a site with different aliases, or when you can want to implement automated certificate handling for a specific certificate authority.

`ssl_options{server_name=ServerName}` Sent back the certificate, key or other ssl options. `ServerName` is the name of the server found in the SSL handshake. Expects a proplist with Erlang `ssl:ssloptions()`. This proplist will override the default ssl options for this connection. For more information about the possible properties see Erlang SSL. When `undefined` is returned the SSL handshake will fail and the connection wil be dropped.

mod_ssl_self_signed

The mod_ssl_self_signed module adds a basic certificate handling functionality to your Zotonic sites. When this module is enabled and there are no certificates or keys available it generates new keys and self signed certificates to get started easily.

SSL Certificates

For a HTTPS connection special encryption keys and certificates are needed. These are supplied by many companies, for very different price ranges, usage and security levels.

There is an exception, when only a secure connection is needed and the only security is against eaves dropping. This is by using a *self signed certificate*. This is a key and certificate that is generated on the server, and does not guarantee anything about the validity of the certificate.

When there is no certificate mod_ssl will generate a self signed certificate.

Certificate and key files

The files with the certificates and key are placed into the `ssl` directory inside the site directory `user/sites/sitename/ssl/`.

Where *sitename* must be replaced with the name of your site.

The files all have the name of the site in them (*sitename* in the filenames below). This is to prevent mixing them up with other sites:

`sitename.pem` This holds the private key for the encryption. The key must be unlocked and in PKCS#1 format (see below).

`sitename.crt` This is the certificate. Usually it is supplied by the certificate authority where you bought it. It can also be a self signed certificate, see below.

`sitename.ca.crt` This is the *CA bundle* that contains root and intermediate certificates for the certificate authority that issued the `sitename.crt` certificate.

The certificate authority will supply these. All supplied certificates are concatenated, with the root certificate last.

The concatenation is a literal command, like:

```
cat intermediate.crt root.crt > sitename.ca.crt
```

This file should not be present when using a self signed certificate.

Dependencies

When mod_ssl needs to generate or convert key and/or certificates it needs **`openssl`**. This program must be installed in the normal search path of the running Zotonic.

Format of the private key

The Erlang SSL implementation uses PKCS#1 format keys. OpenSSL generates (since 2010) PKCS#8 format keys. The difference can be seen when inspecting the key file. A PKCS#1 key starts with:

```
-----BEGIN RSA PRIVATE KEY-----
```

Where a PKCS#8 key starts with:

```
-----BEGIN PRIVATE KEY-----
```

When mod_ssl sees that the key file is a PKCS#8 file then it will stop and give the following error:

```
{error, {need_rsa_private_key, "example.pem", "use: openssl rsa -in sitename.key -
↪out sitename.pem"}}
```

The given command is the command needed to convert the key to a PKCS#1 key. The PKCS#8 key should be renamed to `sitename.key` from `sitename.pem`, before running the above command.

Note that the resulting key file *must* be named `sitename.pem` where *sitename* is the name of the site the key is placed in.

Using SSL certificates

If you order a SSL certificate, the signing authority will ask you which kind of web server you are using and a CSR file. For the web server, select *other*. For the CSR, use the following command (replace `sitename` with the name of your site):

```
openssl req -out sitename.csr -new -newkey rsa:2048 -nodes -keyout sitename.key
```

When OpenSSL asks for the *Common Name* then fill in the site's hostname (e.g. *www.example.com*).

The resulting `.key` file can be converted to a `.pem` file:

```
openssl rsa -in sitename.key -out sitename.pem
```

From the SSL certificate authority you will receive a signed `.crt` file.

See the section *Certificate and key files* above for instructions how to use the `.crt` and `.pem` files.

Generating the self signed certificate

For generating the self signed certificate, mod_ssl runs the following commmands (where `sitename` should be replaced with the name of the site):

```
openssl req -x509 -nodes -days 3650 -subj '/CN=www.example.com' -newkey rsa:2048 \
    -keyout sitename.key -out sitename.crt
```

This generates a private key of 2048 bits and a certificate that is valid for 10 years.

Optionally, when the key turns out to be in PKCS#8 format, mod_ssl will run the following command as well:

```
openssl rsa -in sitename.key -out sitename.pem
```

When the key is already in PKCS#1 format (with older openssl installs) then mod_ssl will rename the `sitename.key` file to `sitename.pem`.

Todo

Add SSL/certificate problem solving.

mod_survey

Adds the concept of *survey resources*: user-definable forms which can be created in the admin interface and filled out by the website's visitors.

Survey question types

The following question types are defined in the survey.

likert Answer a question on a scale of 5 points, from "completely disagree" (1) to "completely agree" (5).

long answer An open question with a big text field.

matching Question type which allows you to match given answers to each other.

narrative Question type for specifying inline questions in a narrative fashion.

page break Breaks the survey into multiple pages.

short answer An open question with a single-lined text field. You have the option of specifying a validation like email, date, numeric.

thurstone A multiple choice field. Like multiple choice, but more powerful. The choices are translatable, and you have the possibility to select either a single answer, multiple answers or submit the form directly when choosing an answer.

true or false Answers a true or false question. You have the option to specify custom texts for both the options.

yes or no Like true or false, answers a true or false question. You have the option to specify custom texts for both the options.

multiple choice A simple multiple choice field that has the added option that the multiple choice can be a numeric value, in which case an overview of the total value will be shown in the printable list and beneath the survey pie chart. This is useful for creating forms which require you to enter an amount or quantity, e.g. for a reservation system. Multiple choice fields cannot currently be translated, use the "thurstone" question type in that case.

category Choose a single resource from a given category as the answer to this question.

subhead Renders a sub-heading between questions.

prompt Renders an extra prompt block.

text block Renders a text block between questions.

Intercepting survey submissions

When a survey is submitted, the survey module sends out a `#survey_submit{}` notification.

This notification has the following fields:

- id - The id of survey being submitted
- handler - A handler name (see below)
- answers - The answers that were filled in
- missing - answers that were missing
- answers_raw - Unprocessed answers, e.g. the raw submission

To intercept a survey submission you would observe this survey_submit notification, and return `ok`:

```
observe_survey_submit(#survey_submit{id=SurveyId}, Context) ->
    ?DEBUG(SurveyId),
    ok.
```

Creating a custom survey handler

The survey edit page has a dropdown for so-called "survey handlers". A survey handler is a property that is set on the resource that indicates the handler that needs to be taken. Handlers are collected using the `#survey_get_handlers{}` *fold* notification.

For instance, the following defines a handler called "email_me":

```
observe_survey_get_handlers(#survey_get_handlers{}, All, Context) ->
  [
   {<<"email_me">>, "E-mail me when survey is submitted"}
   | All
  ].
```

Each handler will show up in the dropdown list and the editor can pick which handler he wants. The value chosen is passed along in the `handler` property of the survey submission, and as such can be used to intercept the survey submission:

```
observe_survey_submit(#survey_submit{handler= <<"email_me">>, id=SurveyId},␣
↪Context) ->
    %% Do something here for surveys which have 'email_me' selected as handler
    ok;
observe_survey_submit(#survey_submit{}, _Context) ->
    %% Let other surveys use the default submision mechanism
    undefined.
```

Todo

Add more documentation

mod_tkvstore

Simple (type,key)/value store. Stores data in the store with minimal latency and (local) serialization of get/put requests.

The store itself is implemented as a PostgreSQL table with columns *type*, *key* and *props*. *props* holds the value for the given type+key combination.

A model, *m_tkvstore* (page 434), is provided to give easy access to the type+key combinations from the templates, and to perform get/put operations from within Erlang.

See also:

m_tkvstore (page 434)

mod_translation

This module provides support for dealing with multiple languages.

How content and static strings are translated is explained in full in *Translation* (page 35).

Language as part of the URL

By default, *mod_translation* (page 317) prefixes each URL (using *URL rewriting* (page 15)) in your website with the code of the current language. The idea behind this is that each language version of a *resource* gets its own URL, and is as such indexable for Google.

This behaviour is enabled by default, but can be switched off in the admin, by going to *Structure*, *Translation*. There is a checkbox at the bottom of the page.

Alternatively you can set the config key `mod_translation.rewrite_url` to `false`.

Programmatically switching languages

In a template, you can use *mod_translation* (page 317)'s *postback* hook to switch between languages:

```
{% button text="Dutch" postback={set_language code="nl"} delegate=`mod_
↪translation` %}
```

Creates a button which switches to Dutch. And another one for english:

```
{% button text="English" postback={set_language code="en"} delegate=`mod_
↪translation` %}
```

Supporting right-to-left languages

To support right-to-left languages like Arabic and Hebrew, you need to make some changes to your templates. mod_translation adds some support to help you.

The idea is to give your `<body/>` tag the text direction of the main item visible on the page. Individual items, for example in menus or context lists, need their own text direction.

You can specify the text direction element attributes by including a template:

```
<body {% include "_language_attrs.tpl" id=id %} >
```

This will generate the following, when Zotonic selected Arabic for the page with id *id*:

```
<body xml:lang="ar" lang="ar" dir="rtl" class="rtl">
```

When you want to add an extra class added to the rtl or ltr class you can use:

```
<body {% include "_language_attrs.tpl" id=id class="my-body-class" %} >
```

And when you want to force a specific language:

```
<body {% include "_language_attrs.tpl" language=`en` %} >
```

mod_twitter

Import Twitter updates in your Zotonic site in realtime.

The Twitter module allows you to have Tweets 'mirrored' from Twitter into your Zotonic site. It does this by creating a background process which has a continuous HTTP connection to the Twitter Streaming API. This way, Tweets arrive instantly at your Zotonic site, no polling necessary!

Installation

Enable the mod_twitter module. Then

In the "config" section in the admin, create 2 new config keys:

`mod_twitter` / `api_login` - Your Twitter username

`mod_twitter` / `api_password` - Your Twitter password

We need your account details to startup the streaming API. mod_twitter will never write to your account: it only reads your updates.

Now, activate the Twitter module under "Modules".

When you have activated the Twitter module, you should go to a person's page in the admin. In the edit page, you see in the sidebar a new item, called "Twitter ID". Here you can enter the numeric Twitter ID of this user. Visit My Twitter Id to find out your twitter ID.

At this point the module will login to Twitter and start following the user(s) that you entered their Twitter IDs for. Only public (or unprotected) Tweets will be imported.

Domain model

The *domain model* for this module is the following: the module creates a *category* (page 5) *Tweet* as a subcategory of *text*. Each time a Tweet is imported, a resource of this category is created.

From the Tweet there is an *author* edge to the person that created the Tweet (i.e. the user who you set the Twitter ID on).

The body text of the Tweet resource is HTML will already be preprocessed with the *twitter* (page 394) filter, so URLs, hash-tags and @-links are converted to HTML already when the Tweet is saved.

The original value of the Tweet (the JSON object received from Twitter) is stored inside the `tweet` property in the Tweet resource, and can be displayed like this:

```
{% print m.rsc[id].tweet %}
```

Using logon with Twitter

Add an app on Twitter and get the consumer key / secret. In Zotonic, configure two config keys, `mod_twitter.consumer_key` and `mod_twitter.consumer_secret` to contain these values. Now set up the callback URL to your Zotonic site in the Twitter app. The logon window will now automatically show a "Sign in with Twitter" button.

See also:

twitter (page 394) filter

mod_video

Adds support for viewing and handling video medium items.

This module converts uploaded videos to h264 and adds a poster (preview) image of the movie.

Note: mod_video uses the command-line utilities `ffmpeg` and `ffprobe`. For mod_video to function correctly they must be present in the search path of Zotonic.

Uploading & conversion

The video module hooks into the media model to intercept any video upload. If a video is uploaded the following steps are done:

- The video is moved to the site's `files/video_queue/` directory.
- A video conversion task is added to the pivot task queue, this task will restart a video conversion in case of any problems.
- A video conversion process is started, supervised by the video module.
- The uploaded medium is replaced by a static `lib/images/processing.png` image.

Only a single video conversion process is allowed to run at any time. This to prevent overloading the server.

After the video is converted the resource's medium record is replaced with the converted video. The frame at 10 seconds (at 1 second for movies shorter than 30 seconds) is added as the preview image of the video.

If a video can't be converted then the video is replaced with the error image, found in `lib/images/broken.png`.

Viewing

The video module extends the `{% media %}` tag for viewing `video/mp4` videos. It uses the template `_video_viewer.tpl` for viewing. For the best viewing results it is best to add `css/video.css` to your included css files.

See also:

mod_video_embed (page 320), *media* (page 453)

mod_video_embed

Adds a tab to the media-upload dialog in the admin which allows the editor to directly paste an embed code from an external site into a media item.

When used in the Zotonic site, the *{% media %}* tag then displays the embed code.

This module is the predecessor to *mod_oembed* (page 307) but still can be used when integrating with services that do not have oEmbed support but do provide HTML embed-code functionality.

See also:

mod_oembed (page 307), *mod_video* (page 319), *media* (page 453)

Todo

Add more documentation

Actions

See also:

Actions (page 25) in the *Templates* (page 19) Developer Guide chapter

Actions

Apply actions with arguments added.

This action takes a list of other actions. One or more arguments are added to the actions before the actions are executed. This action is mostly used in included templates or callbacks. An example can be seen with the action-typeselect action.

Another example, assume we have a template "_list_action.tpl":

```
{% for id in list %}
  <li><a id="{{ #list.id }}" href="#">{{ m.rsc[id].title }}</a></li>
  {% wire id=#list.id action={with_args action=my_action arg={id id}} %}
{% endfor %}
```

Then we can pass an action to this template:

```
{% include "_list_action.tpl" list=[1,2,3,4,5] my_action={redirect dispatch="admin_
↪edit_rsc"} %}
```

The result will be a list of titles for the pages with id 1..5. Every title will be a link to its admin page, as the argument *id* will be added to the *my_action*.

Admin

Action module which provides postback handlers for the "status" view of the admin:

- Rebuild search index
- Flush cache
- Renumber categories

Todo

Extend documentation

Action used in the admin interface for remembering the toggled state of an admin widget.

Todo

Extend documentation

Modules (Admin)

Rescans all modules, to find all templates, lib files, dispatch rules, etc. again.

Todo

Extend documentation

Activate/deactivate a module in the module manager in the admin interface.

Todo

Extend documentation

Backup

Action which starts a manual backup.

Todo

Extend documentation

Config

Trigger the deletion of a configuration value. Used in the admin.

Todo

Extend documentation

Toggle a configuration value. Used in the admin, for instance when displaying a "live" checkbox the state of which should reflect a config value.

Todo

Extend documentation

Open a dialog that asks confirmation to delete a configuration key/value pair.

Todo

Extend documentation

Open a dialog to edit a configuration key/value pair.

Todo

Extend documentation

Open a dialog to create a new configuration key/value pair.

Todo

Extend documentation

Development

Stream template updates to the user agent.

Todo

Dialogs

Opens a dialog with a predefined HTML content and title.

Example:

```
{% button action={dialog title="Wisdom" text="<p>The world is a pancake.</p>"} %}
```

This opens a dialog with the title "Wisdom". The dialog is empty except for the text "The world is a pancake".

Normally, instead of this action, the action action-dialog_open is used. The action action-dialog_open shows a dialog that is rendered on the server.

Argument	Required	Description
title	required	Dialog header title
text	required	Dialog body text
width	optional	Dialog width in pixels
addclass	optional	classname will be appended to default dialog class
backdrop	optional	boolean (0, 1), or the string 'static' for a modal dialog (does not close on backdrop click); default 1
center	optional	boolean (0, 1) default 1; set to 0 to align the dialog at the top

See also:

actions action-dialog_open, action-dialog_close and action-overlay_open.

Renders a template on the server and opens a dialog with the HTML output of the template.

Example:

```
{% button text="cancel" action={dialog_open title="Select a name" template="_
↪select_name.tpl" arg=100} %}
```

The title of this new dialog will be "Select a name", its contents are the output of rendering the template "_select_name.tpl". All arguments are handed as arguments to the template. In this example the template "_select_name.tpl" is rendered with the arguments "title", "template" and "arg".

See also:

actions action-dialog_close, action-dialog and action-overlay_open.

Closes the currently open dialog. When there is no dialog open then nothing happens.

Example:

```
{% button text="cancel" action={dialog_close} %}
```

This button closes any open dialog when clicked.

See also:

actions action-dialog_open and action-dialog.

Renders a template on the server and opens a full screen overlay with the HTML output of the template.

Example:

```
{% button text="show story" action={overlay_open template="_story.tpl" id=1234} %}
```

This opens an overlay over the current content. The template `_story.tpl` will be rendered with the argument `id` (and possibly any other arguments). The rendered html will then be shown inside the overlay.

The overlay template is a `div` with the class `modal-overlay`. Extra classes can be added using the `class` argument:

```
{% wire action={overlay_open template="_splash.tpl" class="splash"} %}
```

See also:

actions action-overlay_close, action-dialog_open and action-dialog.

Closes the currently open overlay. When there is no overlay open then nothing happens.

Example:

```
{% button text="cancel" action={overlay_close} %}
```

This button closes any open overlay when clicked.

See also:

actions action-overlay_open, action-dialog_open and action-dialog.

DOM Elements

Add a css class to an html element.

Example:

```
{% button action={add_class target="myid" class="newclass"} %}
```

Adds the CSS class "newclass" to the element with HTML id "myid".

See also:

actions action-remove_class and action-toggle_class.

Add a `$(..).animate` jQuery call to the target element.

Arguments:

- speed
- easing
- options

Todo

Extend documentation

Todo

Not yet documented.

Add a `$(..).effect` jQuery call to the target element.

Todo

Extend documentation

Show an element by animating the opacity.

Example:

```
{% button action={fade_in target="myid"} %}
```

Shows the element with id “myid” when the button is clicked.

See also:

actions action-toggle, action-show, action-hide, action-fade_out, action-slide_down, action-slide_up, action-slide_fade_in and action-slide_fade_out.

Hide an element by animating the opacity.

Example:

```
{% button action={fade_out target="myid"} %}
```

Hides the element with id “myid” when the button is clicked.

See also:

actions action-toggle, action-show, action-hide, action-fade_in, action-slide_down, action-slide_up, action-slide_fade_in and action-slide_fade_out.

Hide an element without any animation.

Example:

```
{% button action={hide target="myid"} %}
```

Hides the element with id “myid” when the button is clicked.

See also:

actions action-toggle, action-show, action-fade_in, action-fade_out, action-slide_down, action-slide_up, action-slide_fade_in and action-slide_fade_out.

Todo

Not yet documented.

Insert the result of a render action after of an HTML element.

Todo

Extend documentation

Insert the result of a render action before an HTML element.

Todo

Extend documentation

Inserts HTML after the contents of an HTML element.

Adds a template or a literal HTML text after the existing content.

Example:

```
<div id="mydiv"><p>Bye Bye.</p></div>
{% button text="hello" action={insert_bottom target="mydiv" text="<p>Hello World!</
↪p>"} %}
```

After the button is clicked, the contents of the div will be * Bye Bye. Hello World! *.

Another example, now rendering a template:

```
<ul id="mylist"><li>Some item</li></li>
{% button text="hello" action={insert_bottom target="mylist" template="_list_item.
↪tpl" id=42} %}
```

This insert the output of the template *_list_item.tpl* below the existing *
*. All arguments to the update action are also arguments to the template.

See also:

actions action-insert_after, action-insert_before, action-insert_top and action-update.

Inserts HTML before the contents of an HTML element.

Adds a template or a literal HTML text before the existing content.

Example:

```
<div id="mydiv"><p>Bye Bye.</p></div>
{% button text="hello" action={insert_top target="mydiv" text="<p>Hello World!</p>
↪"} %}
```

After the button is clicked, the contents of the div will be * Hello World! Bye Bye. *.

Another example, now rendering a template:

```
<ul id="mylist"><li>Some item</li></li>
{% button text="hello" action={insert_top target="mylist" template="_list_item.tpl
↪" id=42} %}
```

This insert the output of the template *_list_item.tpl* above the existing *
*. All arguments to the update action are also arguments to the template.

See also:

actions action-insert_after, action-insert_before, action-insert_bottom and action-update.

Trigger various jQuery effects on the target element. Mostly, each of these effects have their own action as a shortcut, for example action-show, action-hide.

Arguments:

- type - one of *show*, *hide*, *remove*, *slide_toggle*, *toggle*, *set_class*, *add_class*, *remove_class*, *fade_in*, *fade_out*, *slide_up*, *slide_fade_out*, *slide_fade_in*, *disable*, *enable*, *effect*, *animate*.
- speed
- class
- easing
- effect
- options

Todo

Extend documentation

Places a mask over an element, useful for blocking user interaction during lengthy postbacks.

Example:

```
<a id="{{ #fb_logon }}" href="#facebook"><img src="/lib/images/fb-login-button.png
↪" width="154" height="22" alt="Facebook login button" /></a>
{% wire id=#fb_logon
   action={mask target="logon_outer" message="Waiting for Facebook ."}
   action={redirect dispatch="facebook_authorize"} %}
```

In this example the *logon_outer* div will be masked while the browser is being redirected to Facebook.

Form postbacks are automatically masked.

Note that you need to include a css and a js file to have working masks:

```
{% lib "css/jquery.loadmask.css" %}
{% lib "js/modules/jquery.loadmask.js" %}
```

This action takes three possible arguments:

Argument	Description	Example
target	The id of the element to be masked.	target=”search-form”
message	Message to show next to the spinner image.	message=”Searching...”
delay	Delay (in milliseconds) before the mask is shown. Only shows the mask during lengthy actions.	delay=200

See also:

action action-unmask, action-mask_progress

Sets the progress bar of a action-mask.

The progress bar can be set to a percentage in the range 0...100. The target of the `mask_progress` must be the same as the target of an earlier `mask` action.

Example:

```
{% wire action={mask_progress target="logon_outer" percent=50} %}
```

In this example the *logon_outer* progress bar will show as halfway (50%).

Move an element to another place, appending it to the target. The element is given by id with the `element` argument, or with the `element_sel` argument for a CSS selector.

Todo

Extend documentation

Remove an element from the page.

For example, the following removes the *foo* div from the page:

```
<div id="foo">I am the foo div</div>
{% button text="Remove foo" action={remove target="foo"} %}
```

Without target, the action removes its triggering element:

```
{% button text="Click me to remove me" action={remove} %}
```

See also:

Actions (page 320), scomp-button

Remove a CSS class from an HTML element.

Example:

```
{% button action={remove_class target="myid" class="newclass"} %}
```

Removes the CSS class “newclass” from the element with HTML id “myid”.

See also:

actions action-add_class and action-toggle_class.

Replace the target HTML element by new one.

Todo

Extend documentation

Set the class of an element.

Example:

```
<div id="x" class="not-inited"></div>
{% button text="Init" action={set_class target="x" class="inited"} %}
```

This uses the jQuery *attr('class', class_name)* method to set the new class.

Show an element without any animation.

Example:

```
{% button action={show target="myid"} %}
```

Shows the element with id *myid* when the button is clicked.

See also:

actions action-toggle, action-hide, action-fade_in, action-fade_out, action-slide_down, action-slide_up, action-slide_fade_in and action-slide_fade_out.

Show an element by animating the height.

Example:

```
{% button action={slide_down target="myid"} %}
```

Shows the element with id *myid* when the button is clicked.

See also:

actions action-toggle, action-show, action-hide, action-fade_in, action-fade_out, action-slide_up, action-slide_fade_in and action-slide_fade_out.

Show an element by animating the height and opacity.

Example:

```
{% button action={slide_fade_in target="myid"} %}
```

Shows the element with id *myid* when the button is clicked.

See also:

actions action-toggle, action-show, action-hide, action-fade_in, action-fade_out, action-slide_down, action-slide_up and action-slide_fade_out.

Hide an element by animating the height and opacity.

Example:

```
{% button action={slide_fade_out target="myid"} %}
```

Hides the element with id *myid* when the button is clicked.

See also:

actions action-toggle, action-show, action-hide, action-fade_in, action-fade_out, action-slide_down, action-slide_up and action-slide_fade_in.

Toggle an element by sliding it up and down.

Todo

Extend documentation

Hide an element by animating the height.

Example:

```
{% button action={slide_up target="myid"} %}
```

Hides the element with id *myid* when the button is clicked.

See also:

actions action-toggle, action-show, action-hide, action-fade_in, action-fade_out, action-slide_down, action-slide_fade_in and action-slide_fade_out.

Toggle the visibility of an element.

Example:

```
{% button action={toggle target="myid"} %}
```

Shows the element with id *myid* if it was hidden, otherwise hide it.

See also:

actions action-show, action-hide, action-fade_in, action-fade_out, action-slide_down, action-slide_up, action-slide_fade_in and action-slide_fade_out.

Toggle a CSS class from an HTML element.

Example:

```
{% button action={toggle_class target="myid" class="newclass"} %}
```

When the HTML element with id “myid” has the CSS class “newclass” then it is removed, otherwise it is added.

See also:

actions action-add_class and action-remove_class.

Removes a mask that was placed over an element using the action-mask action.

Example:

```
{% wire action={unmask target="logon_outer"} %}
```

In this example the mask over the *logon_outer* div will be removed.

See also:

action action-mask.

Updates the content of an HTML element with a template or a literal HTML text.

Example:

```
<div id="mydiv"><p>Bye Bye.</p></div>
{% button text="hello" action={update target="mydiv" text="<p>Hello World!</p>"} %}
```

When clicked, the contents of the div will be set to the HTML fragment * Hello World! *. This replaces any content present.

Another example, now rendering a template:

```
<ul id="mylist"><li>Some item</li></li>
{% button text="hello" action={update target="mylist" template="_list_item.tpl"␣
↪id=42} %}
```

This updates the *
* with the output of the template *_list_item.tpl*. All arguments to the update action are also arguments to the template.

Note: Use the action-update_iframe action for updating the contents of an `iframe` element.

Argument	Description	Example
target	The id of the element receiving the rendered HTML.	*target="myview"*
text	Literal HTML text to be inserted, no escaping will be done.	*text="Hello World"*
template	Name of the template to be rendered.	*template="_list_view.tpl"*
include_all	Add this argument to include all templates with the same name. If not added then the best template will be used.	*include_all*
catinclude	Add this argument to use a *catinclude* (page 444) instead of a normal include of the template. The *id* argument *must* be present for a catinclude to work.	*catinclude id=1*

All other arguments are passed as-is to the included template(s).

See also:

actions action-update_iframe, action-insert_top and action-insert_bottom.

Updates the content of an iframe with a template or a literal HTML text.

Note: This action is only used to update an `iframe` element. Use the action-update action for updating the contents of a normal HTML element.

Example:

```
<iframe id="preview"></iframe>
{% button text="Show Email" action={update_iframe target="preview" template="email.
↪tpl" recipient_id=m.acl.user} %}
```

When clicked, the contents of the iframe will be set to the rendered `email.tpl` template. This replaces any content present.

Argument	Description	Example
target	The id of the iframe receiving the rendered HTML document.	*target="myview"*
text	Literal HTML doc to be inserted, no escaping will be done.	*text="<html>...</html>"*
template	Name of the template to be rendered.	*template="page.tpl"*
catinclude	Add this argument to use a *catinclude* (page 444) instead of a normal include of the template. The *id* argument *must* be present for a catinclude to work.	*catinclude id=1*

All other arguments are passed as-is to the included template(s).

See also:

action action-update.

Editor

Add WYSIWYG editor controls to all textarea's with the `z_editor` class in the target.

Todo

Extend documentation

Remove any WYSIWYG editor controls from all textarea's with the `z_editor` class in the target.

Todo

Extend documentation

Used for inserting an internal link in the TinyMCE editor in the admin.

Todo

Extend documentation

Used for triggering the insertion of a media item in the TinyMCE editor in the admin.

Todo

Extend documentation

Used after a media item is selected in the media chooser for the TinyMCE editor.

Todo

Extend documentation

Used by the admin as a callback when a media file has been selected for insertion into the rich-text editor.

Todo

Extend documentation

Events

New in version 0.8.

Send a zotonic notify message. All modules which observe this message are notified.

Example:

```
{% button action={notify message=`clicked`} %}
```

Sends the message *clicked* to the notify system. All modules which are subscribed to the *clicked* message are notified.

This action sends a message to the event handler on the server.

Example:

```
{% button text="Go" action={postback postback="go" action={growl text="sent message
↪"}} %}
```

Note: The scomp-button scomp can also take a postback argument directly.

After clicking the button the event *go* will be sent to the *controller* module on the server and a action-growl message will be displayed.

The *event/2* function in the controller module will be called as:

```
event({postback, go, TriggerId, TargetId}, Context)
```

This action can have the following arguments:

Argument	Description	Example
postback	The message that will be send to the server module.	postback={my_message arg="hello"}
delegate	The name of the Erlang module that will be called. Defaults to the controller module generating the page.	delegate="my_module"
action	Any other actions that will be executed when the postback is done. This parameter can be repeated.	action={show target="wait"}
inject_args	If set to *true*, and postback is a tuple (as in the *my_message* example in this table), any values from the args in the postback will replace the arg value in the postback argument. This is useful when the arg is coming from an outer action and not set explicitly in the template code (as is done in the example for illustration). The value of *some_arg* in the postback handler will be *123*.	{postback postback={my_event some_arg} inject_args some_arg=123}
qarg	Post the value of an input or select with the postback. The value of the *qarg* argument is the id of the element to be posted. Multiple *qarg* arguments can be given. On the server the value will be available as a normal query argument using *z_context:get_q/2*	qarg="my-input-id"

New in version 0.9.0: Added *inject_args* option.

Trigger a named {% wire %} with an action. All args will be args to the named wire. The trigger's `name` argument is the name of the wire.

Todo

Extend documentation

Forms

Sets the “disabled” attribute of a HTML tag and adds the CSS class “disabled”.

Example:

```
<input id="myid" type="text" value="hello" />
{% button text="disable" action={disable target="myid"} %}
```

After clicking the button the input will be:

```
<input id="myid" disabled="disabled" class="disabled" type="text" value="hello" />
```

See also:

action action-enable.

Resets the “disabled” attribute of a HTML tag and removes the CSS class “disabled”.

Example:

```
<input id="myid" disabled="disabled" class="disabled" type="text" value="hello" />
{% button text="enable" action={enable target="myid"} %}
```

After clicking the button the input will be:

```
<input id="myid" class="" type="text" value="hello" />
```

See also:

action action-disable.

Bind actions to a jQuery event or submit a form.

This action is the base action for the scomp-wire scomp. Normally this event is not used directly.

Add a `$(..).focus()` jQuery call to the target element to give it input focus.

Resets the target form to its initial state.

Todo

Extend documentation

Resets the enclosing form, a specifically targeted form or the closest form to an element.

Example:

```
<form method="get" action="/search">
    <input type="text" name="q" value="" />
    {% button text="search" action={submit} %}
    {% button text="cancel" action={reset} %}
</form>
```

Another example:

```
<form id="search-form" method="get" action="/search">
   <input type="text" id="q" name="q" value="" />
</form>
{% button text="search" action={submit closest="q"}
{% button text="cancel" action={reset closest="q"} %}
```

Clicking on the button will reset the form *search-form* as it is the closest form to the element with id *q*.

The reset form action is mostly used in the result of event handlers.

This action takes two possible arguments, when neither is defined then the form enclosing the trigger element will be reset.

Argument	Description	Example
target	The id of the form to be reset.	target="search-form"
closest	The id of an element that is close to the form to be reset. When no argument value is supplied then it defaults to the id of the trigger element (for example the button the action is coupled to).	closest

Set the value of a form field.

Example:

```
<input type="text" id="x" name="xyz" value="" />
{% button text="fill" action={set_value target="x" value="etaoinstrdlu"} %}
```

Clicking on the button will set the value of the input element to the most interesting string *etaoinstrdlu*.

This action can set the value of any input element, select or text area. It uses the jQuery *val()* method to set the value.

Submits the enclosing form, a specifically targeted form or the closest form to an element.

Example:

```
<form method="get" action="/search">
  <input type="text" name="q" value="" />
  {% button text="search" action={submit} %}
</form>
```

Another example:

```
<form id="search-form" method="get" action="/search">
  <input type="text" id="q" name="q" value="" />
</form>
{% button text="search" action={submit closest="q"} %}
```

Clicking on the button will submit the form *search-form* as it is the closest form to the element with id *q*.

The submit action is mostly used in the result of event handlers.

This action takes two possible arguments, when neither is defined then the form enclosing the trigger element will be submitted.

Argument	Description	Example
target	The id of the form to be submitted.	target="search-form"
closest	The id of an element that is close to the form to be submitted. When no argument value is supplied then it defaults to the id of the trigger element (for example the button the action is coupled to).	closest

Show possible selections whilst typing.

Performs a search for the typed text whilst typing in an input field. Shows possible matching pages in a selectable list.

Example:

```
<form method="get" action="/search">
  <input type="text" id="person" name="person" value="" />
  <ul id="suggestions"></ul>
```

```
  <input type="hidden" id="person_id" value="" />
  {% wire id="person" type="keyup"
          action={typeselect cat="person"
                             target="suggestions"
                             action_with_id={with_args action={set_value target=
↪"person_id"} arg={value select_id}}
                             action={submit}}
  %}
</form>
```

This is a rather complicated example. It connects the typeahead action to the input element. The list of suggestions will be shown in the *
* with id *suggestions*. Only pages in the category *person* will be found.

The listed suggestions will have two actions attached. One action will set the value of the hidden input element *person_id* to the id of the selected suggestion (which is a *page*). The other action will submit the form.

The *action_with_id* arguments are always performed before the *action* arguments.

The *typeselect* action accepts the following arguments:

Argument	Description	Example
target	The id of element that will show the list of suggestions.	target="mylist"
cat	The category for the searched pages. This argument can be repeated.	cat="text"
template	Template used to show the list of possible pages. This defaults to the template "_action_typeselect_result.tpl". The template gets the following arguments: result (list of ids), action_with_id and action.	template="_show_suggestions.tpl"
action_with_id	Actions executed when a suggestion is selected. The id of the selected page will be added as the *id* parameter. This argument can be repeated.	action_with_id={postback postback="page_select"}
action	Actions executed when a suggestion is selected. This list is executed after the *action_with_id* actions. This argument can be repeated.	action={slide_up target="form-id"}

Render a validation error on the target. Text is given in the *text* argument.

Todo

Extend documentation

JavaScript

This action executes JavaScript directly. It can be used to interface with non-Zotonic JavaScript libraries and functions.

Example:

```
{% button text="hello" action={script script="alert('hello world')"} %}
```

Clicking on the button will show a JavaScript alert with the text *hello world* in it.

Using template variables:

```
{% with "world" as recipient %}
    {% button
        text="hello"
        action={
            script
            script="alert('hello " ++ recipient ++ "')"
        }
```

```
    %}
{% endwith %}
```

Log

Internal action used by *mod_logging* (page 298) to allow realtime updates of the log view.

Todo

Extend documentation

Mailing list

Shows the dialog to mail the current page (*resource*) to a single e-mail address. This is used in the frontend of a site, for instance in *mod_base_site* (page 287) to "share" the current page over e-mail.

Todo

Extend documentation

Shows the dialog to mail the current page (*resource*) to a mailing list. This is used in the admin "mailing status" interface.

Todo

Extend documentation

Post a message to the test mailing list, given with the `id` argument.

The `on_success` argument decides which actions are triggered after the page has been sent.

Todo

Extend documentation

Confirm a mailinglist subscription. Required argument is the `confirm_key`.

Other arguments:

- `on_success` - actions which get executed when the subscription is confirmed.
- `on_error` - actions which get executed when the subscription fails (e.g. wrong confirm key).

Todo

Extend documentation

Cancel a mailing list subscription. The recipient id is given with the `id` argument.

The `on_success` argument decides which actions are triggered after unsubscribe is successful; `on_error` actions are triggered when unsubscribe fails.

Todo

Extend documentation

Notifications

Show an alert dialog.

Example:

```
{% button action={alert text="hello world"} %}
```

Shows an alert dialog with the text “hello world”.

Alert accepts the following arguments:

Argument	Description	Example
title	Title of the alert.	title=”Alert”
text	The text to be displayed.	text=”Hola!”
button	Text for the button. Defaults to “OK”	button=”Bummer”
only_text	Set this to not show the “OK” button.	only_text
action	Action to be done when the user clicks on the OK button. There can be multiple actions.	

The alert dialog is rendered using the `_action_dialog_alert.tpl` template. Overrule this template to change the contents of the alert dialog.

See also:

actions action-growl and action-confirm.

Show a javascript confirm message and on confirmation triggers one or more actions and/or send a postback to the server.

Example:

```
{% button action={confirm text="Format hard disk?" action={growl text="Better not"}
↪} %}
```

Shows a javascript dialog with the question “Format hard disk?”. When this dialog is confirmed then the growl message “Better not” will appear. When the dialog is denied or canceled then nothing happens.

When there is a postback defined then the event handler for the postback will be called like:

```
event(#postback{message=Message, trigger=TriggerId, target=TargetId}, Context).
```

Confirm accepts the following arguments:

Argument	Description	Example
text	The text to be displayed.	text=_"The answer to life and the rest?"
title	Title above the alert, defaults to _"Confirm"	title=_"Rescue the world, with an answer"
ok	The text of the ok button, defaults to _"OK"	text="42"
cancel	The text of the cancel button, defaults to _"Cancel"	text="No, thanks for the fish"
text_template	Template used to render the text, all action arguments are passed to the template.	text_template="_fancy_confirm.tpl"
action	One or more actions to be executed on confirmation. This argument can be repeated.	action={alert text="you said ok"}
on_cancel	One or more actions to be executed on cancelation	on_cancel={alert text="you said cancel"}
postback	Event to be sent back to the server if the ok button is clicked.	postback="clicked_confirm"
delegate	Erlang module handling the postback. Defaults to the controller generating the page.	delegate="my_event_module"
is_danger	If the 'ok' button should be flagged as dangerous.	is_danger

See also:

actions action-alert and action-growl.

Show a message in the upper right corner of the browser window. The message will automatically disappear after some time.

Example:

```
{% button action={growl text="hello world"} %}
```

Shows a message with the text "hello world".

Growl accepts the following arguments:

Argument	Description	Example
text	The text to be displayed.	text="Hola!"
stay	When true then the message does not disappear automatically	stay
type	Type of the message, one of "notice" or "error". Default is "notice".	type="error"

See also:

actions action-alert and action-confirm; and *Enabling Growl Notifications* (page 216).

Page handling

This action redirects the browser to another page or back to the previous page.

Example:

```
{% button text="home" action={redirect location="/"} %}
```

Redirects back to the home page when the button is clicked.

Back in history example:

```
{% button text="Back" action={redirect back} %}
```

After clicking the button the browser will go back to the last page using the JavaScript history.

Example of using dispatch rules for the redirect location:

```
{% button text="edit" action={redirect dispatch="admin_edit_rsc" id=my_id} %}
```

When clicked the browser is redirected to the admin edit page for the *resource* with the id of *my_id*.

This action can have the following arguments:

Argument	Description	Example
back	When given then the browser is directed to the previous page.	back
dispatch	The name of a dispatch rule. All other parameters are assumed to be parameters for the dispatch rule.	dispatch="admin"
id	When back and dispatch are not defined then the redirect uri will be the page_url of the resource.	id=42
location	The http address to redirect to. Can be an url with or without host name.	location="http://example.com"

Reload the current page.

Example:

```
{% button text="refresh" action={reload} %}
```

Clicking on the button will reload the page.

Predicates and Connections

Show a dialog for creating a new *predicate*.

Todo

Extend documentation

Add an *edge* between two *resources*. Used in the admin.

The edge is selected with either:

- the argument `edge_id`
- the arguments `subject_id`, `predicate`, `object_id`

For instance:

```
{% button
    text="Add"
    class="btn"
    action={
        link
        subject_id=id
        predicate="contains"
        object_id=other_id
        action={
            reload
        }
    }
%}
```

Other arguments:

- element_id
- edge_template
- action - actions executed after linking

See also:

action-unlink

Todo

Extend documentation

Remove an *edge* between two *resources*. Used in the admin.

The edge is selected with either:

- the argument `edge_id`
- the arguments `subject_id`, `predicate`, `object_id`

For instance:

```
{% button
    text="Remove"
    class="btn"
    action={
        unlink
        subject_id=id
        predicate="contains"
        object_id=other_id
        action={
            reload
        }
    }
%}
```

Other arguments:

- hide - selector to fade out after unlink
- edge_template - passed on to the undo action template
- action - actions executed after unlink
- undo_action - passed on to the undo action template
- undo_message_id - defaults to *unlink-undo-message*

After update, an undo message is rendered in the *undo_message_id* target, with the template `_action_unlink_undo.tpl`.

See also:

action-link

Todo

Extend documentation

Resources

Delete a media file from a *resource*, without confirmation.

Todo

Extend documentation

Delete a *resource*, without confirmation.

Todo

Extend documentation

Open a dialog to confirm the deletion of a *resource*.

Todo

Extend documentation

Open a dialog to duplicate the current *resource* with a new id and title.

Todo

Extend documentation

Open a dialog to edit the "basic" information of a *resource*.

The basic information usually comprises of the title, the summary and the category, but what exactly is displayed as "basic" info is dependent on the *category* of the resource and can be changed per category by making a category specific template named `_admin_edit_basics_form.tpl` which is included using a *catinclude* (page 444).

For instance, to create a special "basics" dialog for the category *news*, you would create a template called `_admin_edit_basics_form.news.tpl`

Todo

Extend documentation

Shows the admin dialog for uploading a media item. See *Media* (page 23).

Todo

Extend documentation

Show the admin dialog for creating a new *resource*.

When the resource is created, the user is redirected to the admin edit page. This action exports a postback called `new_page` which is used to create the page:

```
{% wire id=#form type="submit"
      postback={new_page subject_id=subject_id predicate=predicate␣
↪redirect=redirect
                        actions=actions callback=callback}
      delegate=`action_admin_dialog_new_rsc`
%}
```

This postback has the following arguments:

- `subject_id` + `predicate`: Create an edge from the given subject to this new page, using the given predicate.
- `redirect`: Boolean flag whether or not to redirect to the edit page. Defaults to `true`.
- `actions`: Any actions to perform after the resource is created.
- `callback`: JavaScript function to call when the subject edge has been created.
- `objects`: A list of `[object, predicate]` pairs which are created as outgoing edges from the new page to the given objects. The object can be a resource ID or a resource name. Example:

```
objects=[ [m.acl.user, "author"] ]
```

creates an "author" edge from the new page to the currently logged in user.

Todo

Extend documentation

Search

Show more results of the current search query inline on the page.

The *moreresults* action is an alternative to using a next/previous pager to paginate through search results. Instead, *moreresults* lets you load more results from the current search, directly onto the same page. This feature is similar to Twitter's *more* button, Slashdot's *many more* button, and others.

Using it is quite simple. The only special thing you need is that every result item should go into its own template. The minimal example is something like the following:

```
{% with m.search[{query cat="media" pagelen=10 }] as result %}
  <div id="results">
    {% for id in result %}
      {% include "_item.tpl" %}
    {% endfor %}
  </div>

  {% button text="more..." action={moreresults result=result
                                   target="results"
                                   template="_item.tpl"}
  %}
{% endwith %}
```

The *moreresults* action has a *result* argument, which should point to the search result that you want to show more of. The *target* attribute denotes the container that the new items get appended to, and the *template* argument shows which template needs to be appended.

The number of items that get added is equal to the *pagelen* setting of the search.

When there are no more items, the *moreresults* button will get disabled automatically.

Normally the template is called for every row in the search result. This is useful for lists. Sometimes all results must be rendered together, for example when special grouping is needed. In those case the argument *is_result_render* must be added. Example:

```
{% with m.search[{query cat="media" pagelen=16 }] as result %}
  <div id="results">
     {% include "_items.tpl" %}
  </div>

  {% lazy action={moreresults result=result
                              target="results"
                              template="_items.tpl"
                              is_result_render
                              visible}
  %}
{% endwith %}
```

Note that here we use the *lazy* scomp, which will perform the action if it is scrolled into view. Because we are using the *lazy* scomp we have to add the *visible* argument so that the re-loaded *moreresults* action will be wired for visibility and not on for a click. In this way the page loads automatically more results if the user is scrolls down.

Where *_items.tpl* displays the found pages in rows of four elements:

```
{% for ids in result|chunk:4 %}
   <div class="row-fluid">
   {% for id in ids %}
       <div class="span4">
           <h3><a href="{{ id.page_url }}">{{ id.title }}</a></h3>
           <p>{{ id.summary }}</p>
       </div>
   {% endfor %}
   </div>
{% empty %}
   <div class="row-fluid"></div>
{% endfor %}
```

Sessions

Set a session variable.

Example:

```
{% button action={session_set key="foo" value="bar"} action={reload} %}
```

This sets the session variable “foo” to the value “bar”, and then reloads the page. In your templates, you can use *m_session* (page 433) to retrieve this value again:

```
{{ m.session.foo }}
```

`key` and `value` are both required values.

Set a persistent variable.

Example:

```
{% button action={persistent_set key="foo" value="bar"} %}
```

This sets the session variable “foo” to the value “bar”. In your templates, you can use *m_persistent* (page 423) to retrieve this value again:

```
{{ m.persistent.foo }}
```

`key` and `value` are both required values.

Signals

The connect action allows one to attach other actions to a specified signal. When the signal is emitted, the specified actions are performed.

Example:

```
{connect signal={signal_type prop1=value} action={...}}
```

At-tributes	Description
signal	The pattern of a signal to which you want to connect to.
action	The action you want to perform when a signal matching the pattern is emitted.
name	The name of the connection. Giving a connection a name makes it possible to disconnect it at a later stage.

Disconnect from a named slot

Example:

```
{disconnect name="my-signal"}
```

Attributes	Description
name	Disconnect from the named slot.

Emit a signal.

Example:

```
{emit signal={signal_type prop1=123}}
```

Attributes	Description
signal	The signal you want to emit. When there is a slot which matches the pattern of the emitted signal, it will receive the signal. Note that it is possible that multiple slots match this signal.

Templates

Render a template. When used in a postback action, the result will be sent back with the response data for the postback.

This is useful when you want to send the output from a template back as response in a postback or submit event handler.

Example:

```
z_render:wire({template, [{template, "my_response.tpl"}, {data,
Response}]}, Context).
```

Template accepts the following arguments:

Argument	Description	Example
template	Name of template to render.	template="my_template.tpl"
•	Any other arguments will be passed on to the template being rendered.	id=123

User

Todo

Not yet documented.

Delete the username from a user, no confirmation.

Todo

Extend documentation

Open a dialog to confirm the deletion of the username of a user.

Todo

Extend documentation

Show a dialog for setting a username / password on the given *resource* (which is usually a person).

Todo

Extend documentation

Show a dialog for adding a user. This creates a *person* *resource* and adds a username / password to it.

Todo

Extend documentation

This action logs off the current user and reloads the current page as the anonymous visitor.

Example:

```
{% button text="Log off" action={logoff} %}
```

After clicking the button the page will reload and the current user will be signed out.

Controllers

This is the full list of *controllers* that are available in Zotonic. For more general information about controllers, see the *Controllers* (page 11) manual.

controller_admin

- Module: *mod_admin* (page 275)

The admin controller is the main controller behind which admin pages are served. Its main purpose is that it does an authentication check (Is current user allowed to `use` the module `mod_admin`).

The *template* parameter decides which admin template gets served, and defaults to *admin.tpl*.

Todo

Extend documentation

controller_admin_acl_rules_export

- Module: *mod_acl_user_groups* (page 271)

Todo

Not yet documented.

controller_admin_backup

- Module: *mod_backup* (page 286)

Shows the admin backup screen where you can download nightly backups that were made by *mod_backup* (page 286).

Todo

Extend documentation

controller_admin_backup_revision

- Module: *mod_backup* (page 286)

Shows the admin backup revisions screen where you can see older version for a *resource*.

Todo

Extend documentation

controller_admin_category_sorter

- Module: *mod_admin_category* (page 279)

Shows the admin category screen where you can edit the *category* tree, rearranging the categories, adding new categories, or removing existing ones.

Todo

Extend documentation

controller_admin_comments

- Module: *mod_comment* (page 287)

Shows an admin screen with an overview of most recently created comments. The screen offers the option to moderate the comments or delete them entirely.

Todo

Extend documentation

controller_admin_comments_settings

- Module: *mod_comment* (page 287)

Shows an admin settings screen where you can edit settings related to *mod_comment* (page 287).

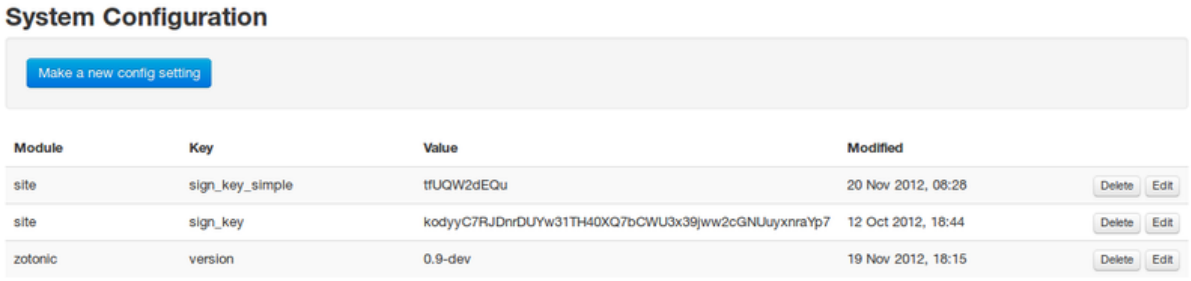
Todo

Extend documentation

controller_admin_config

- Module: *mod_admin_config* (page 279)

Shows the admin config editor. Here you can edit the key/value pairs of *m_config* (page 415).



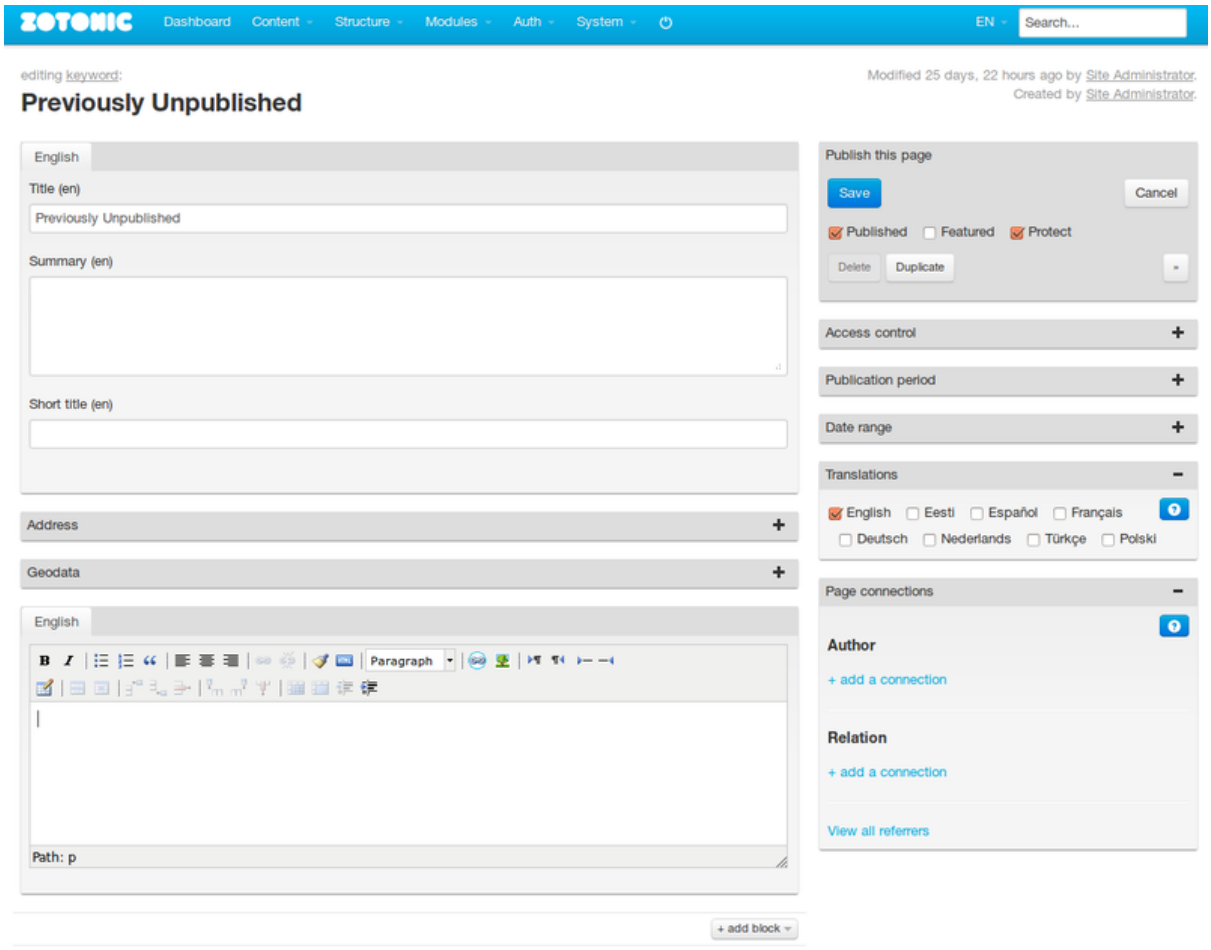
Todo

Extend documentation

controller_admin_edit

- Module: *mod_admin* (page 275)

The main admin edit controller. This controller serves the edit page where *resources* can be edited.



Todo

Extend documentation

controller_admin_mailing_preview

- Module: *mod_mailinglist* (page 299)

This controller shows a preview of what a resource that is being mailed would look like, in a popup window.

Todo

Extend documentation

controller_admin_mailing_status

- Module: *mod_mailinglist* (page 299)

This controller shows the mailing status of a *resource*. It lists each mailing list available in the system, and shows whether or not the current resource has already been sent to the list.

Per mailinglist, it offers the options to send the resource right now, or schedule it for later delivery.

There are also buttons for sending the resource to a test mailing list or to a single email address.

Todo

Extend documentation

controller_admin_mailinglist

- Module: *mod_mailinglist* (page 299)

This controller shows the mailing lists that are available in the system.

For each list, it shows the number of recipients and the title. Clicking a list shows the *recipients* (page 348) of the mailing list.

Todo

Extend documentation

controller_admin_mailinglist_recipients

- Module: *mod_mailinglist* (page 299)

Shows the recipients of the current mailing list. The recipients are listed in three columns, and have a checkbox next to them to deactivate them.

Clicking a recipient shows a popup with information about the recipient, where you can edit the e-mail address and the recipient's name details.

The page also offers buttons for importing and exporting lists of email addresses.

Todo

Extend documentation

controller_admin_media_preview

- Module: *mod_admin* (page 275)

A controller for rendering preview thumbnails of any media embedded in a richtext-editor component of a *resource* on the *admin edit controller* (page 346) page.

Todo

Extend documentation

controller_admin_module_manager

- Module: *mod_admin_modules* (page 283)

Shows the list of Zotonic modules currently known to the system.

The list is sorted based on the module's status: active modules are listed first, non-active modules next.

Each module has a button which let you toggle the active status of the module.

Todo

Extend documentation

controller_admin_referrers

- Module: *mod_admin* (page 275)

Shows the list of pages (*resources*) which refer to this *resource* through an *edge*.

Todo

Extend documentation

controller_admin_seo

- Module: *mod_seo* (page 310)

Shows a form with settings related to Search Engine Optimization.

Todo

Extend documentation

controller_api

- Module: *mod_base* (page 287)

`controler_api` processes authorized REST API requests: It provides an easy way to create API calls to allow computer programs to perform functions on your Zotonic site.

`controller_api` by default intercepts all URLs according to the patterns `/api/:module/:method` and the URL `/api/:module`. See the *API services* (page 51) manual for more information.

Authentication

See *Service authentication* (page 55) on how authentication is done when using this controller.

Creating services at a non-standard URL

New in version 0.8.

It is possible to pre-fill the required *module* and *method* parameters so that you can use `controller_api` at another entry point. For instance, the following *dispatch rule* (page 12) is valid:

```
{dosomething,    ["do", "something"], controller_api, [{module, "foobar"}, {method,
↪ "test"}]}
```

This would invoke the `mod_foobar/services/service_foobar_test.erl` service at the url */do/something*,

See also:

Dispatch rules (page 12) and *Controllers* (page 11).

controller_atom_entry

- Module: *mod_atom* (page 286)

Renders a representation of the given *resource* as Atom XML.

Todo

Extend documentation

---

controller_atom_feed_cat

- Module: *mod_atom_feed* (page 286)

Renders an *Atom XML feed* based on the given *category*.

Todo

Extend documentation

---

controller_atom_feed_search

- Module: *mod_atom_feed* (page 286)

Renders an *Atom XML feed* based on the given search terms.

For which search can be used, see the *Search* (page 27) document.

Todo

Extend documentation

---

controller_close_connection

- Module: *mod_base* (page 287)

Closes the browser connection. Used primarily to work around an apparent Safari bug.

Todo

Extend documentation

controller_comet

- Module: *mod_base* (page 287)

Controller which is used by the scomp-stream tag which keeps a connection open to the browser for browsers that do not support WebSockets, to use long-polling as fallback.

The comet controller is used to transport data from the server to the browser.

See *Transport* (page 49) for more information about transporting data between the server and the browser.

Todo

Extend documentation

controller_connection_test

- Module: *mod_base* (page 287)

Todo

Not yet documented.

controller_cookies

- Module: *mod_base* (page 287)

Todo

Not yet documented.

controller_export

- Module: *mod_export* (page 293)

Todo

Not yet documented.

controller_export_resource

- Module: *mod_export* (page 293)

Todo

Not yet documented.

controller_facebook_authorize

- Module: *mod_facebook* (page 293)

Controller which redirects the user to the authorize uri of Facebook, to let the user login to the website with their Facebook account.

Todo

Extend documentation

controller_facebook_redirect

- Module: *mod_facebook* (page 293)

Thic controller handles the OAuth redirect of the Facebook logon handshake, when the user has authorized with Facebook and returns to the site.

See http://developers.facebook.com/docs/authentication/

Todo

Extend documentation

controller_file

- Module: *mod_base* (page 287)

Serve an uploaded-, resized- or library file.

This controller is used to serve files and images. It is able to manipulate an image according to the parameters supplied.

Image manipulation parameters are signed to prevent random image manipulations on the request of visitors, which might result in a denial of service due to processing- or disk space limitations.

This controller serves all files with a very long client side caching time and handles if-modified-since checks. Text files are served with gzip compression if the user-agent supports it.

Multiple files can be served in a single request; the controller concatenates them into a single file. See the *lib* (page 453) tag for more information. The creators of the files have to ensure that they can be properly concatenated.

Dispatch rules and options

Example dispatch rules:

```
{image, ["image", '*'], controller_file, []},
{lib, ["lib", '*'], controller_file, [{root, [lib]}]}
```

controller_file has the following dispatch options:

Option	Description	Example
root	List of root directories where files are located. Use 'lib' for the library files. This defaults to the site's "files/archive" directory.	{root, [lib]}
path	Default file to be served. Used for files like "robots.txt" and "favicon.ico".	{path,"misc/robots.txt"}
content_disposition	If the file should be viewed in the browser or downloaded. Possible values are `inline` and `attachment`. Defaults to the browser's defaults by not setting the "Content-Disposition" response header.	{content_disposition, inline}
acl	Extra authorization checks to be performed.	See *ACL options* (page 352).
max_age	Max age, used for Cache and Expires. Value is an integer, number of secs.	{max_age,3600}

ACL options

Authorization (page 38) checks to perform, in addition to the `acl_action` dispatch option, can be given in the `acl` dispatch option, and accepts the following options:

ACL option	Description	Example
`is_auth`	Disable anonymous access to this resource.	`{acl, is_auth}`
`logoff`	Log out user before processing the request.	`{acl, logoff}`
`{Action, Resource}`	Check if user is allowed to perform `Action` on `Resource`. The example is equivalent to the options `{acl_action, edit}, {id, my_named_page}`.	`{acl, {edit, my_named_page}}`
`[{Action, Resource}]`	A list of checks to be performed, as above.	`{acl, [ {view, secret_page}, {update, 345} ]}`
`ignore`	Don't perform any access control checks. Be careful to add your own checks in the rendered template and all its included templates.	`{acl, ignore}`

More about the search root

The search root can be a list with one or more of the following:

- The atom *lib* for finding library files in the *lib* directory of modules.
- The atom *template* for finding files in the *template* directory of modules.
- A directory name (binary or string). This directory name must be absolute or relative to the *files* directory of the site.
- A tuple *{module, ModuleName}* to refer to a module. The module must implement the functions *file_exists/2* and *file_forbidden/2*.

CSS and JavaScript templates

If a file with a *lib* or *template* root is not found, then the same filename with the addition of *.tpl* is checked. For example *styles.css.tpl*. If found then the template will be rendered against an empty site context. This means that,

with the current implementation, the template will not receive the current language, user etc. This behavior may change in the future.

Note: `controller_file` replaces `controller_file_readonly` and `controller_lib`

See also:

controller_file_id (page 353), *lib* (page 453), *image* (page 449), *image_url* (page 451)

New in version 0.11.

controller_file_id

- Module: *mod_base* (page 287)

Redirect to the controller controller_file.

This controller maps a resource id to the filename of the medium associated with the resource.

For the redirect it uses the dispatch rule defined in the dispatch options.

Examples from mod_base:

```
{media_inline, ["media","inline","id",id], controller_file_id, [ {dispatch, media_
↪inline}, {ssl, any} ]},
{media_inline, ["media","inline",'*'], controller_file, [ {content_disposition,␣
↪inline}, {ssl, any} ]},
```

The first dispatch rule will redirect to the second. If no associated file was found, then a 404 is returned.

See also:

controller_file (page 352)

controller_http_error

- Module: *mod_base* (page 287)

Todo

Not yet documented.

controller_id

- Module: *mod_base* (page 287)

Handle different content representations of a page.

Redirects to different representations of a page, depending on the requested content type. The redirect is done using a "303 See Other" status. A "404 Not Found" or "410 Gone" is returned if the requested page never existed or has been deleted.

When no content types are requested then `text/html` is selected.

This controller is also used for a page's short url representation.

Example dispatch rule (from mod_base):

```
{id, ["id", id], controller_id, []}
```

This controller does not have any dispatch options.

This controller handles the following query argument:

Option	Description	Example URL
id	Id of the requested *resource*.	/id/1234

The list of provided content types is collected with a *foldr* notification (see *Notifications* (page 46)) of the type `content_types_dispatch`. Modules should add their provided content types in front of the accumulator. The added entries are tuples: `{MimeType, DispatchRuleName}`.

Example of adding a content type handler adding a text/plain handler with the dispatch rule `rsc_text`:

```
observe_content_types_provided(content_types_dispatch, Acc, Context) ->
    [{"text/plain", rsc_text} | Acc].
```

controller_instagram_authorize

- Module: *mod_instagram* (page 298)

Todo

Not yet documented.

controller_instagram_push

- Module: *mod_instagram* (page 298)

Todo

Not yet documented.

controller_instagram_redirect

- Module: *mod_instagram* (page 298)

Todo

Not yet documented.

controller_language_set

- Module: *mod_translation* (page 317)

Controller which sets the language as given in the `code` argument, and redirects the user back to the page given in the `p` argument.

Todo

Extend documentation

controller_linkedin_authorize

- Module: *mod_linkedin* (page 298)

Todo

Not yet documented.

controller_linkedin_redirect

- Module: *mod_linkedin* (page 298)

Todo

Not yet documented.

controller_logoff

- Module: *mod_authentication* (page 286)

Controller that logs off a user, destroying the session. It also removes any "remember me" cookies the user has, so that auto-logon is disabled.

Todo

Extend documentation

See also:

controller_logon (page 356), *Authentication* (page 37).

controller_logon

- Module: *mod_authentication* (page 286)

This controller logs on a user, and optionally sets a "remember me" cookie.

When `p` argument is given, the user is redirect to the page given.

The controller also has postback `event/2` calls for the following interactions:

- Login confirmation
- Password reset
- Send password reminder

Todo

Extend documentation

See also:

controller_logoff (page 356), *Authentication* (page 37).

controller_mailinglist_export

- Module: *mod_mailinglist* (page 299)

Controller which downloads the given mailinglist id as a CSV file.

Todo

Extend documentation

controller_oauth_access_token

- Module: *mod_oauth* (page 306)

Controller implementing the *exchange request token for access token* action in the OAuth 1.0 flow.

http://oauth.net/core/1.0/#auth_step3

Todo

Extend documentation

controller_oauth_apps

- Module: *mod_oauth* (page 306)

Admin controller which shows an admin screen with the list of currently registered OAuth apps (*consumers*).

Clicking an app shows the details of it: its name, tokens, and on a second tab the permissions are listed that users that the app can get once users authorize it.

Todo

Extend documentation

controller_oauth_authorize

- Module: *mod_oauth* (page 306)

Controller implementing the *authorization* action in the OAuth 1.0 flow, which prompts the user to allow the application access to certain *services*.

http://oauth.net/core/1.0/#auth_step2

Todo

Extend documentation

controller_oauth_request_token

- Module: *mod_oauth* (page 306)

Controller implementing the *request token* action in the OAuth 1.0 flow.

http://oauth.net/core/1.0/#auth_step1

Todo

Extend documentation

controller_page

- Module: *mod_base* (page 287)

Show a rsc as a HTML page.

This controller is used to show the HTML page of a *resource*. A "404 Not Found" or "410 Gone" page is shown if the requested page never existed or has been deleted.

The user will be redirected to the `logon` URL when the current user is not allowed to view the page.

This controller also adds a `noindex` response header when the page's "seo_noindex" flag is set.

Example dispatch rule:

```
{page, ["page", id], controller_page, []}
```

Dispatch arguments

`controller_page` recognizes the following arguments inside the dispatch pattern:

Argument	Description	Example URL
id	The id of the page (rsc) to be shown. This can be the numerical id or the unique name of a page.	/page/12345

Dispatch options

The following options can be given to the dispatch rule:

Option	Description	Example
id	Id or unique name of the resource to be shown. This overrules any id in the query arguments.	{id, page_about}
template	Name of the template to be rendered. Defaults to "page.tpl" Can also be a tuple of the following form: *{cat, Name}*. See also: *cat-include* (page 444).	{template, "about.tpl"} {template, {cat, "home. tpl"}}
cat	The category the resource that is requested has to be. If a page of a different category is requested, a 404 is shown.	{cat, text}
acl_action	What ACL action will be checked. Defaults to 'view'; but can also be 'edit' if users need edit permission on the rsc to be able to access the resource.	{acl_action, edit}
acl	Extra authorization checks to be performed.	See *ACL options* (page 358).
is_canonical	Whether this URL should be considered the caninical URL of the page. If so, the controller will redirect to the sc's page path if set. Defaults to true.	{is_canonical, false}

ACL options

Authorization (page 38) checks to perform, in addition to the `acl_action` dispatch option, can be given in the `acl` dispatch option, and accepts the following options:

ACL option	Description	Example
`is_auth`	Disable anonymous access to this resource.	`{acl, is_auth}`
`logoff`	Log out user before processing the request.	`{acl, logoff}`
`{Action, Resource}`	Check if user is allowed to perform `Action` on `Resource`. The example is equivalent to the options `{acl_action, edit}`, `{id, my_named_page}`.	`{acl, {edit, my_named_page}}`
`[{Action, Resource}]`	A list of checks to be performed, as above.	`{acl, [ {view, secret_page}, {update, 345} ]}`
`ignore`	Don't perform any access control checks. Be careful to add your own checks in the rendered template and all its included templates.	`{acl, ignore}`

See also:

controller_template (page 362).

controller_postback

- Module: *mod_base* (page 287)

The postback controller is the endpoint for AJAX callbacks and transports from the browser to the server.

See *Transport* (page 49) for more information about transporting data between the server and the browser.

This controller is used internally by scomp-wire and action-postback.

HTML form posts

It is possible to directly post HTML forms to the postback controller.

The method of the form must be POST, the action has two possibilities:

```
<form method="POST" action="/postback/mymessage">...</form>
```

or:

```
<form method="POST" action="/postback/mymessage/mymodule">...</form>
```

The first example will call:

```
z_notifier:first(#submit{message="mymessage"}, Context).
```

The second example will call the mentioned module directly:

```
MyModule:event(#submit{message="mymessage"}, Context).
```

All posted query arguments are available via the usual `z_context:get_q("arg", Context)` calls.

Actions and other generated javascript can only returned to the browser if the `z_page_id` was included in the post. Body content and headers can be set in the *Context* and will be sent back to the browser.

Todo

Extend documentation

---

controller_redirect

- Module: *mod_base* (page 287)

Redirect to another url.

This controller redirects a request to another URL. The URL can be a fixed URL, the location of a fixed page id or the name of a dispatch rule.

Example dispatch rule using the redirect controller:

```
{redir, ["plop"], controller_redirect, [{url, "/newplop"}, {is_permanent, true}]}
```

This redirects any requests of "/plop" permanently to "/newplop".

It has the following dispatch options:

Option	Description	Example
url	The url of the new location the browser is sent to.	{url, "/example"}
dispatch	Name of a dispatch rule to use for the location url. All arguments (except dispatch and is_permanent) are used as parameters for the dispatch rule.	{dispatch, admin}
id	Id of the page to redirect to. The controller will redirect to the page_url of this id. The id can be an integer or the name of the page (use an atom or a binary).	{id, 123}
qargs	A list with querystring arguments to use in the new dispatch rule. Specifies what query (or dispatch) arguments to use from this dispatch rule into the dispatch rule that is being redirected to.	{qargs, [id, slug]}
is_permanent	Use a permanent (301) or temporary redirect (307). Defaults to false.	{is_permanent, false}

This controller does only handle request arguments that are specifically noted in the "qargs" list (and then only when the "dispatch" argument is set).

Example

A dispatch rule that always redirects /foo/12312/slug to /bar/12312/slug:

```
{bar, ["bar", id, slug], controller_page, [{template, "bar.tpl"}]},
{bar_redirect, ["foo", id, slug], controller_redirect, [{dispatch, bar}, {qargs,␣
↪[id,slug]}]}
```

controller_rest_rsc

- Module: *mod_rest* (page 308)

This controller implements a RESTful endpoint for *resources* and its *m_rsc* (page 426) model.

Todo

Extend documentation

controller_signup

- Module: *mod_signup* (page 311)

Controller which displays a form to sign up (rendered from `signup.tpl`).

It also implements the nessecary postbacks to perform the signup and log a user in.

Todo

Extend documentation

controller_signup_confirm

- Module: *mod_signup* (page 311)

Controller which displays the confirmation page where the user can confirm his signup.

The template used is `signup_confirm.tpl`.

Todo

controller_static_pages

- Module: *mod_base* (page 287)

Serve a static page or pages.

With this controller it is possible to add a folder with static files as a sub-site to your Zotonic site. Add the folder and all files to a directory in your template directory or your site's directory and define the directory in a dispatch rule.

Example dispatch rule:

```
{oldsite, ["old", '*'], controller_static_pages, [{root, "old_site"}]}
```

When a file `a.txt` is requested this resource will check for `a.txt` and `a.txt.tpl`. When it finds a `.tpl` file then that file be handled as a template. All dispatch configuration variables are available in the template.

Directories will be redirected to the directory name with a `/` appended. The resource serves the file `index.html` or `index.html.tpl` for the directory contents. If these are not found, it will give a 404 page, unless the `allow_directory_index` option is set; in which case a directory listing is displayed.

It has the following dispatch options:

Option	Description	Example
root	Name of the directory in the site directory containing the static files. The root is a path name relative to the current site's base directory.	{root, "oldsite"}
use_cache	Whether or not served files are cached in memory for an hour. Defaults to false. Use this for high-volume traffic when the files themselves do not change often.	{use_cache, true}
allow_directory_index	Whether or not to serve a directory listing when no index file is found. Defaults to false. The directory index is rendered using template-directory_index. New in version 0.9.	{allow_directory_index, true}

This resource does not handle any request arguments.

controller_template

- Module: *mod_base* (page 287)

Show a template.

This controller renders the template configured in the dispatch rules.

Example dispatch rule:

```
{home, [], controller_template, [{template, "home.tpl"}]}
```

This will render the `home.tpl` template at the url `/`.

Dispatch arguments

`controller_template` recognizes the following arguments inside the dispatch pattern:

Argument	Description	Example URL
id	A resource id to be used in the template. This can be the numerical id or the unique name of a page. More commonly the id is given as a dispatch option.	/page/12345

Dispatch options

The following options can be given to the dispatch rule:

Option	Description	Example
template	Name of the template to be rendered. Can also be a tuple of the following form: *{cat, Name}*. See also: *catinclude* (page 444).	{template, "home.tpl"} {template, {cat, "home.tpl"}}
anonymous	Render the template always as the anonymous user, even when an user is logged on. Defaults to false.	{anonymous, true}
content_type	The content type provided by the dispatch rule. Defaults to "text/html".	{content_type, "application/json"}
maxage	The number of seconds of how long to cache this file in the browser. Sets the response header: *Cache-control: public; maxage=X*.	{maxage, 3600}
acl_action	What ACL action will be checked. Defaults to 'view'; but can also be 'edit' if users need edit permission on the rsc to be able to access the resource.	{acl_action, edit}
acl	Extra authorization checks to be performed.	See *ACL options* (page 363).
id	Id or unique name of a resource to be referenced in the rendered template. This overrules and id from the query arguments.	{id, page_about}

ACL options

Authorization (page 38) checks to perform, in addition to the `acl_action` dispatch option, can be given in the `acl` dispatch option, and accepts the following options:

ACL option	Description	Example
`is_auth`	Disable anonymous access to this resource.	`{acl, is_auth}`
`logoff`	Log out user before processing the request.	`{acl, logoff}`
`{Action, Resource}`	Check if user is allowed to perform `Action` on `Resource`. The example is equivalent to the options `{acl_action, edit}, {id, my_named_page}`.	`{acl, {edit, my_named_page}}`
`[{Action, Resource}]`	A list of checks to be performed, as above.	`{acl, [ {view, secret_page}, {update, 345} ]}`
`ignore`	Don't perform any access control checks. Be careful to add your own checks in the rendered template and all its included templates.	`{acl, ignore}`

See also:

controller_page (page 357).

controller_twitter_authorize

- Module: *mod_twitter* (page 318)

Controller which redirects the user to the authorize uri of Twitter, to let the user login to the website with their Twitter account.

Todo

Extend documentation

controller_twitter_redirect

- Module: *mod_twitter* (page 318)

Thic controller handles the OAuth redirect of the Twitter logon handshake, when the user has authorized with Twitter and returns to the site.

Todo

Extend documentation

controller_unload_beacon

- Module: *mod_base* (page 287)

Todo

Not yet documented.

controller_user_agent_probe

- Module: *mod_base* (page 287)

Controller which serves a small JavaScript file which is use to determine the user agent of the browser, for properties which cannot be determined from the User Agent header, like the screen size and (multi)touch-capability.

The JavaScript also communicates the timezone back to the server. This timezone is then stored in the persistent cookie and the session.

Todo

Extend documentation

controller_user_agent_select

- Module: *mod_base* (page 287)

Controller which changes the selected user agent preference for the current session, by letting the user choose from a form or list of links.

Todo

Extend documentation

controller_website_redirect

- Module: *mod_base* (page 287)

This controller does a redirect to the `website` property of the given *resource*.

Todo

Extend documentation

controller_websocket

- Module: *mod_base* (page 287)

Controller which accepts a WebSocket connection from the browser.

The controller provides persistent WebSocket connections between the client and the server. Since Zotonic 0.11, a WebSocket connection is automatically started on the page (unless nostream is given in the script tag).

See *Transport* (page 49) for more information about transporting data between the server and the browser.

Defining Custom Websocket Behaviour

You can provide your own websocket_start similar too controller websocket by setting a ws_handler containing the name of a websocket handler module in the zotonic context.

Example:

```
websocket_start(ReqData, Context) ->
    Context1 = z_context:set(ws_handler, ?MODULE, Context),
    controller_websocket:websocket_start(ReqData, Context).
```

When passing a custom handler module, the default handler websocket will not be used, but the specified one. Controller websocket contains the code for the default zotonic handler. It attaches itself as websocket handler to the page session.

It is also possible to configure a custom `ws_handler` by specifying it in a dispatch rule.:

```
{customws, ["socket", "custom"], controller_websocket, [{ws_handler, my_ws_handler}
↪]}
```

WebSocket Handler API

In order to implement your own websocket handler you have to implement four callback functions. When you want to sent a message to the client you call `controller_websocket:send_data/2`.

Example:

```
-module(my_ws_handler).

-export([websocket_init/1,
         websocket_message/3,
         websocket_info/2,
         websocket_terminate/2]).

%% @doc Called when the websocket is initialized.
websocket_init(_Context) ->
    erlang:send_after(1000, self(), <<"Hello!">>),
    ok.

%% @doc Called when a message arrives on the websocket.
websocket_message(Msg, From, Context) ->
    controller_websocket:websocket_send_data(From, ["You said: ", Msg]).

%% @doc Called when another type of message arrives.
websocket_info(Msg, _Context) ->
    controller_websocket:websocket_send_data(self(), Msg),
    erlang:send_after(5000, self(), <<"Hello again!">>).

%% @doc Called when the websocket terminates.
websocket_terminate(Reason, Context) ->
    ok.
```

The websocket_init, websocket_info and websocket_terminate callbacks are called from within the controller's receive loop, so to send a message to the websocket, you send it to `self()`, as in the example above.

The *websocket_message* function however gets a *From* argument passed to it because it is called from another process. To send a message to the socket, you need to send it to the *From* pid.

controller_wmtrace

- Module: *mod_development* (page 289)

Admin controller which display traces of webmachine requests.

Todo

Extend documentation

controller_wmtrace_conf

- Module: *mod_development* (page 289)

Admin controller which let you configure and enable/disable the display of traces of webmachine requests.

Todo

Extend documentation

All dispatch rules

All the dispatch rules from all modules. For a background on dispatch rules, see *The URL dispatch system* (page 12).

Filters

Binaries

first

- Module: *mod_base* (page 287)

Returns the first character or element.

Returns the first byte of a binary or the first element of a list. An empty binary is returned when the input is empty.

For example:

```
{{ value|first }}
```

If the value is `hello` then the output is `h`.

Note: This function is safe to use with multibyte character values, if the input is a binary.

For a regular list:

```
{{ [1,2,3]|first }}
```

The filtered value is `1`.

See also:

tail (page 390), *last* (page 367)

last

- Module: *mod_base* (page 287)

Returns the last character or element.

Returns the last element of the value. When the value is a list then the last element of the list is returned, when the value is a binary then the last *byte* of the binary is returned.

For example:

```
{{ value|last }}
```

When value is the list `hello` then the output will be `o`.

Note: This function is not safe to use with multibyte character values, use with care.

See also:

first (page 366)

length

- Module: *mod_base* (page 287)

Returns the length of the value.

The length of a list is the number of elements in the list, the length of a binary is the number of bytes in the binary.

For example:

```
{{ value|length }}
```

When value is the list "hello" then the output will be "5".

Note: **With multi-byte values this function does not return the** number of characters, it returns the number of bytes. This may change in a future release.

to_binary

- Module: *mod_base* (page 287)

Convert the input to a binary value.

Example:

```
{{ "Hello"|to_binary }}
```

Results in the binary value `<<"Hello">>`.

This filter uses the `z_convert:to_binary/1` function.

See also:

stringify (page 404)

Booleans

yesno

- Module: *mod_base* (page 287)

Show a boolean value as a text.

Given a string mapping values for `true`, `false` and (optionally) `undefined`, returns one of those strings according to the value.

Non-empty values are converted to their boolean value first using `z_convert:to_boolean/1`.

Example:

```
{{ 1|yesno:"ja,nee" }}
```

Will output "ja"; this:

```
{{ 0|yesno:"ja,nee" }}
```

Will output “nee”.

yesno accepts these values:

Value	Argument	Output
true	“yeah,no,maybe”	“yeah”
false	“yeah,no,maybe”	“no”
undefined	“yeah,no,maybe”	“maybe”
undefined	“yeah,no”	“no”

Dates

add_day

- Module: *mod_base* (page 287)

Adds a day to a date. The value must be of the form `{{Y,M,D},{H,I,S}}`.

For example:

```
{{ value|add_day }}
```

When the value is `{{2008,12,10},{15,30,0}}`, the output is `{{2008,12,11},{15,30,0}}`.

The filter has an optional argument which defines the number of days to add:

```
{{ value|add_day:3 }}
```

When the value is `{{2008,12,10},{15,30,0}}`, the output is `{{2008,12,13},{15,30,0}}`.

See also:

sub_day (page 373), *add_week* (page 369), *add_month* (page 369), *add_year* (page 369)

add_month

- Module: *mod_base* (page 287)

Adds a month to a date. The value must be of the form `{{Y,M,D},{H,I,S}}`.

For example:

```
{{ value|add_month }}
```

When the value is `{{2008,12,10},{15,30,0}}`, the output is `{{2009,1,10},{15,30,0}}`.

The filter has an optional argument which defines the number of months to add:

```
{{ value|add_month:3 }}
```

When the value is `{{2008,12,10},{15,30,0}}`, the output is `{{2009,3,10},{15,30,0}}`.

add_week

- Module: *mod_base* (page 287)

Adds a week to a date. The value must be of the form `{{Y,M,D},{H,I,S}}`.

For example:

```
{{ value|add_week }}
```

When the value is `{{2008,12,10},{15,30,0}}`, the output is `{{2008,12,17},{15,30,0}}`.

The filter has an optional argument which defines the number of weeks to add:

```
{{ value|add_week:4 }}
```

When the value is `{{2008,12,10},{15,30,0}}`, the output is `{{2009,1,7},{15,30,0}}`.

add_year

- Module: *mod_base* (page 287)

Adds a year to a date. The value must be of the form `{{Y,M,D},{H,I,S}}`.

For example:

```
{{ value|add_year }}
```

When the value is `{{2008,12,10},{15,30,0}}`, the output is `{{2009,12,10},{15,30,0}}`.

The filter has an optional argument which defines the number of years to add:

```
{{ value|add_year:3 }}
```

When the value is `{{2008,12,10},{15,30,0}}`, the output is `{{2011,12,10},{15,30,0}}`.

date

- Module: *mod_base* (page 287)

Formats a date or datetime according to the format specified in the argument.

The date should be a tuple `{Y,M,D}` and the datetime should be a tuple `{{Y,M,D},{H,I,S}}`. Dates and datetimes are always assumed to be in *local time*.

An example:

```
{{ mydate|date:"Y-m-d" }}
```

If mydate is `{2009,6,1}` this returns `2009-06-01` as output.

To show the year of the current date:

```
{{ now|date:"Y" }}
```

See also the *timesince* (page 375) filter to display a human readable *relative* time like *10 hours ago*.

Timezones

Dates in Zotonic are stored in UTC. If a date is displayed then it is converted to the timezone of the current request context. This timezone can be one of the following, in order of preference:

- Preferred timezone set by the user
- Timezone of the user-agent
- Default timezone of the site
- Default timezone of the Zotonic server
- UTC

A specific timezone can be enforced by adding a second parameter to the date-filter. For example, to display a date in UTC:

```
{{ mydate|date:"Y-m-d H:i T":"UTC" }}
```

Timezone and *all day* date ranges

If a resource's date range is set with the *date_is_all_day* flag then the dates are not converted to or from UTC but stored as-is. This needs to be taken into account when displaying those dates, otherwise a conversion from (assumed) UTC to the current timezone is performed and the wrong date might be displayed.

The timezone conversion can be prevented by adding the *date_is_all_day* flag to the date-filter as the timezone. Example, for displaying the start date of a resource:

```
{{ id.date_start|date:"Y-m-d":id.date_is_all_day }}
```

Date formatting characters

Date uses the same format as PHP's date function with some extensions and some omissions.

All supported formatting characters are listed below:

Character	Description
a	"a.m." or "p.m." (note that this follows Associated Press style and adds periods).
A	Uppercase "AM" or "PM".
b	Month, textual, in three lowercase characters.
c	ISO-8601 date format.
d	Day of the month in two digits with leading zero, i.e. "01" to "31".
D	Day of the week, textual, three letters of which the first one uppercase.
f	If minutes is zero then show only the hour, otherwise the hour and the minutes. Hours are shown using the "g" form
F	Month, textual, full english name with first character in uppercase.
g	12 Hour format without leading zero, i.e. "1" to "12".
G	24 Hour format without leading zero, i.e. "0" to "23".
h	12 Hour format with leading zero, i.e. "01" to "12".
H	24 Hour format with leading zero, i.e. "00" to "23".
i	Minutes with leading zero, i.e. "00" to "59".
i	Daylight saving time flag. "1" if DST is in effect, "0" if no DST.
j	Day of the month without leading zero, i.e. "1" to "31".
l	(lowercase L) Day of the week, textual, full english name with first character in uppercase.
L	Boolean for whether the year is a leap year. Returns the string "True" or "False".
m	Month with leading zero, i.e. "01" to "12".
M	Month, textual, in three characters, first character in uppercase.
n	Month without leading zero, i.e. "1" to "12".
N	Month abbreviation in Associated Press style. March, April, June and July are shown in full. September as "Sept." a
O	Difference to Greenwich Mean Time (GMT).
P	Time in 12 hour format with minutes and "a.m." or "p.m." appended. Minutes are left off if they are zero, and the st
r	RFC 2822 formatted date.
s	Seconds with leading zero.
S	English ordinal suffix for the day of the month, 2 characters; i.e. "st", "nd", "rd" or "th".
t	Number of days in the given month, i.e. "28" to "31".
T	Timezone used for displaying the date.
U	Seconds since the Unix epoch of January 1, 00:00:00 GMT.
w	Day of the week, numeric. 0 For sunday to 6 for saturday.
W	ISO-8601 week number of the year, starting on mondays.

Table 5.1 – continued fro

Character	Description
y	Year in two digits.
Y	Year in four digits.
z	Day of the year, i.e. 1 to 366.

To construct a date in a template, the filter also accepts Erlang lists as input, so the following will work:

```
{{ [1990,10,10]|date:"j F Y" }}
```

Will output *10 October 1990*. This also works with datetimes:

```
{{ [[1990,10,10],[10,11,12]]|date:"j F Y - H:i:s" }}
```

Will output *10 October 1990 - 10:11:12*.

See also:

date_range (page 371), *datediff* (page 372), *timesince* (page 375), *now* (page 454)

date_range

- Module: *mod_base* (page 287)

Show a date range.

Filter to simplify displaying datetime ranges. When displaying a datetime range, the display of the dates and times often depends if the date parts of the datetimes are equal or not.

Take the following code:

```
{{ [fromdate, todate]|date_range:[format_ne, sep, format_eq] }}
```

If the `dates` of fromdate and todate are equal then the output will be as if the following were written:

```
{{ fromdate|date:format_ne }}{{ sep }}{{ todate|date:format_eq }}
```

However, if the dates are equal then the output will be as if the following were written:

```
{{ fromdate|date:format_ne }}{{ sep }}{{ todate|date:format_ne }}
```

See also:

date (page 369)

datediff

- Module: *mod_base* (page 287)

Calculate the difference between two dates, returning a single part of that difference.

The filter takes a list with 2 parts *[start, end]* as date range argument.

The filter argument the "part" that will be extracted, and is one of *Y*, *M*, *D*, *H*, *I*, *S*.

Example, where start = 2012-02-02, end = 2012-03-01:

```
{{ [end, start]|datediff:"M" }}
```

Returns *1*, since the difference in months between those 2 dates is 1.

See also:

date (page 369)

eq_day

- Module: *mod_base* (page 287)

Tests if the value is a date and equal to the argument. The value and the argument must be a tuple of the format `{Y,M,D}` or `{{Y,M,D},{H,I,S}}`.

For example:

```
{% if value|eq_day:othervalue %}same day{% endif %}
```

This outputs "same day" if value and othervalue are dates and on the same day.

This is useful for conditions, in combination with for example the *if* tag.

See also:

ne_day (page 373)

in_future

- Module: *mod_base* (page 287)

Tests if a date is in the future.

Tests if the value is a date and in the future. The value must be a tuple of the format `{Y,M,D}` or `{{Y,M,D},{H,I,S}}`. When the value is not a date, or datetime, the result will be `undefined`.

For example:

```
{% if value|in_future %}That day has yet to come.{% endif %}
```

This outputs "That day has yet to come." if the value is a date and in the future.

See also:

in_past (page 373)

in_past

- Module: *mod_base* (page 287)

Tests if a date is in the past.

Tests if the value is a date and in the past. The value must be a tuple of the format `{Y,M,D}` or `{{Y,M,D},{H,I,S}}`. When the value is not a date or datetime, the result is `undefined`.

For example:

```
{% if value|in_past %}Those days have gone.{% endif %}
```

This outputs "Those days have gone." if the value is a date and in the past.

See also:

in_future (page 372)

ne_day

- Module: *mod_base* (page 287)

Tests if two dates are not equal.

Tests if the value is a date and not equal to the argument. The value and the argument must be a tuple of the format `{Y,M,D}` or `{{Y,M,D},{H,I,S}}`.

For example:

```
{% if value|ne_day:othervalue %}different days{% endif %}
```

This outputs "different days" if value and othervalue are dates and different.

This is useful in combination with for example the if tag.

See also:

eq_day (page 372)

sub_day

- Module: *mod_base* (page 287)

Subtracts a day from a date. The value must be of the form `{{Y,M,D},{H,I,S}}`.

For example:

```
{{ value|sub_day }}
```

When the value is `{{2008,12,10},{15,30,0}}` then the output is `{{2008,12,9},{15,30,0}}`.

The filter has an optional argument which defines the number of days to subtract:

For example:

```
{{ value|sub_day:3 }}
```

When the value is `{{2008,12,10},{15,30,0}}` then the output is `{{2008,12,7},{15,30,0}}`.

See also:

add_day (page 368), *sub_week* (page 374), *sub_month* (page 374), *sub_year* (page 374)

sub_month

- Module: *mod_base* (page 287)

Subtracts a month from a date. The value must be of the form `{{Y,M,D},{H,I,S}}`.

For example:

```
{{ value|sub_month }}
```

When the value is `{{2008,12,10},{15,30,0}}` then the output is `{{2008,11,10},{15,30,0}}`.

The filter has an optional argument which defines the number of months to subtract:

For example:

```
{{ value|sub_month:3 }}
```

When the value is `{{2008,12,10},{15,30,0}}` then the output is `{{2008,12,7},{15,30,0}}`.

See also:

sub_day (page 373), *sub_week* (page 374), *add_month* (page 369), *sub_year* (page 374)

sub_week

- Module: *mod_base* (page 287)

Subtracts a week from a date. The value must be of the form `{{Y,M,D},{H,I,S}}`.

For example:

```
{{ value|sub_week }}
```

When the value is `{{2008,12,10},{15,30,0}}` then the output is `{{2008,12,3},{15,30,0}}`.

The filter has an optional argument which defines the number of weeks to subtract:

For example:

```
{{ value|sub_week:3 }}
```

When the value is `{{2008,12,10},{15,30,0}}` then the output is `{{2008,11,19},{15,30,0}}`.

See also:

sub_day (page 373), *add_week* (page 369), *sub_month* (page 374), *sub_year* (page 374)

sub_year

- Module: *mod_base* (page 287)

Subtracts a year from a date. The value must be of the form `{{Y,M,D},{H,I,S}}`.

For example:

```
{{ value|sub_year }}
```

When the value is `{{2008,12,10},{15,30,0}}` then the output is `{{2007,12,10},{15,30,0}}`.

The filter has an optional argument which defines the number of years to subtract:

For example:

```
{{ value|sub_year:3 }}
```

When the value is `{{2008,12,10},{15,30,0}}` then the output is `{{2005,12,10},{15,30,0}}`.

See also:

sub_day (page 373), *sub_week* (page 374), *sub_month* (page 374), *add_year* (page 369)

timesince

- Module: *mod_base* (page 287)

Show a readable version of a date/time difference.

Translates the difference between two dates into a simple readable string like “2 minutes, 10 seconds ago”.

Optionally takes an argument with the date to compare against, which is by default the current local date/time.

Example:

```
{{ my_date|timesince }}
```

When "my_date" is `{{2008,12,10},{15,30,0}}` and the current date/time is `{{2009,11,4},{13,50,0}}` then this outputs "10 months, 24 days ago". When the time value is in the future then it outputs a string like "in X minutes".

This function does not take daylight saving changes into account.

Extra arguments

The `timesince` filter can take several extra arguments, in the order of arguments:

- Base date to use. Useful to show the difference between two dates, defaults to `now`
- Text to use for the relative time designations, defaults to `"ago,now,in"`
- Format for the printout. Now two options `1` and `2`, for the number of components shown. For example `2` will show *2 minutes, 10 seconds ago* where `1` will show *2 minutes ago*

Example

Show the time between creation and modification of a resource:

```
{{ id.created|timesince:id.modified:"":1 }}
```

This might display something like:

```
10 days
```

See also:

date (page 369), *now* (page 454)

utc

- Module: *mod_base* (page 287)

Translates a datetime from local time to UTC.

For example:

```
{{ id.modified|utc|date:"Ymd:His\\Z" }}
```

Displays the modification date and time of a resource in Universal Time.

See also:

date (page 369)

Encryption

md5

- Module: *mod_base* (page 287)

Translates a string to a md5 hex value.

For example:

```
{{ "The quick brown fox jumps over the lazy dog"|md5 }}
```

Creates:

```
9E107D9D372BB6826BD81D3542A419D6
```

Note that MD5 is not considered to be a very safe encryption algorithm.

sha1

- Module: *mod_base* (page 287)

Translate a string to a sha1 hex value.

This filter creates a SHA-1 checksum of the input string. It is output as a hex digest:

```
{{ "Hello world"|sha1 }}
```

Outputs the value: "7B502C3A1F48C8609AE212CDFB639DEE39673F5E"

Character escaping

brlinebreaks

- Module: *mod_base* (page 287)

Translate HTML `<br/>` elements into ASCII newlines (`\n`).

The following string:

```
{{ "foo<br/>bar"|brlinebreaks }}
```

will evaluate to `foo\nbar`.

Note: Non-closing line breaks (`<br>`) are currently not converted.

See also:

linebreaksbr (page 379)

escape

- Module: *mod_base* (page 287)

HTML escape a text. Escapes all reserved HTML characters in the value. Escaped strings are safe to be displayed in a HTML page. When you echo a query string argument or path variable then you must escape the value before displaying it on a HTML page.

The following characters are replaced:

Character	Replacement
>	`>`
<	`<`
"	`&quot;`
'	`&#039;`
&	`&`

The escaping is only applied if the filter is not within an `{% autoescape on %}` block. If you always want escaping to be applied, use the *force_escape* (page 379) filter.

For example:

```
{{ value|escape }}
```

When the value is `<hel&lo>` then the output is `&lt;hel&amp;lo&gt;`.

Note: this filter is not part of a module, it is built into ErlyDTL.

See also:

force_escape (page 379)

escape_ical

- Module: *mod_base* (page 287)

Escape the value according to the RFC2445 rules.

A double quote becomes `\"`; a comma becomes `\,`; a colon becomes `":"`; a semicolon becomes `\;`; a backslash becomes `\\` and a newline becomes `\n`.

It is also ensures that any single line is maximum 70 characters long by splitting the lines with newline/space combinations.

For example:

```
{{ value|escape_ical }}
```

When the value is `abc:d;e` then the output is `abc":"d\;e`.

See also:

escape (page 377)

escape_link

- Module: *mod_base* (page 287)

Convert any URLs in a plaintext into HTML links, with adding the `rel="nofollow"` attribute.

Example:

```
{{ "http://foo.bar/"|escape_link }}
```

Outputs:

```
<a href="http://foo.bar/" rel="nofollow">http://foo.bar/</a>
```

This filter is very useful when displaying user-generated plaintexts, like comments.

See also:

urlize (page 382)

escapejs

- Module: *mod_base* (page 287)

Escapes the value for insertion in JavaScript output.

For example:

```
{{ value|escapejs }}
```

When the value is `he'llo` then the output is `he\x27llo`.

Internally, this calls `z_utils:js_escape/1` to perform the escaping.

Note: when generating JSON output, be sure to use *escapejson* (page 378), as JSON escaping is subtly different from JS escaping.

See also:

escape (page 377), *escapejson* (page 378)

escapejson

- Module: *mod_base* (page 287)

Escapes the value for insertion in JSON output.

For example:

```
{{ value|escapejson }}
```

When the value is `he'llo` then the output is `he\x27llo`.

Internally, this calls `z_utils:json_escape/1` to perform the escaping.

See also:

escape (page 377), *escapejs* (page 378)

escapexml

- Module: *mod_base* (page 287)

Escape the value for insertion in xml output.

For example:

```
{{ value|escapexml }}
```

When the value is `<hel'lo>` then the output is `&#60;hel&#39;lo&#62;`.

fix_ampersands

- Module: *mod_base* (page 287)

Replaces ampersands in the value with "&" entities.

For example:

```
{{ value|fix_ampersands }}
```

If the value is `hel&lo` then the output is `hel&lo`.

See also:

escape (page 377)

force_escape

- Module: *mod_base* (page 287)

HTML escapes a text.

Applies HTML escaping to a string (see the *escape* (page 377) filter for details). In contrary to the *escape* filter, the *force_escape* filter is applied *immediately* and returns a new, escaped string. This is useful in the rare cases where you need multiple escaping or want to apply other filters to the escaped results. Normally, you want to use the *escape* (page 377) filter.

For example:

```
{{ value|force_escape }}
```

If the value is `hel&lo` then the output is `hel&lo`.

See also:

escape (page 377)

linebreaksbr

- Module: *mod_base* (page 287)

Translate ASCII newlines (`\n`) into HTML `<br />` elements.

The following string:

```
{{ "foo\nbar"|linebreaksbr }}
```

will evaluate to `foo<br />bar`.

See also:

brlinebreaks (page 376)

slugify

- Module: *mod_base* (page 287)

Converts a text into a slug.

Makes the value safe for use as a part in an url. Mostly used for adding titles or descriptions to an url.

For example:

```
{{ value|slugify }}
```

When value is "Nichts is unmöglich!" then the output will be "nichts-is-unmoglich".

See also:

stringify (page 404)

unescape

- Module: *mod_base* (page 287)

Removes HTML escaping from a text.

Expands the entities added by the *escape* (page 377) filter or *force_escape* (page 379) filter. This is useful when you want to display a field from the database in a text-only format medium.

For example:

```
Title: {{ m.rsc[id].title|unescape }}
```

Be careful that you only use this filter when you are absolutely sure that the output is not used in HTML or XML.

See also:

escape (page 377), *force_escape* (page 379)

urlencode

- Module: *mod_base* (page 287)

Make a text safe for URLs.

Translates all url unsafe characters in the value to their percent encoding.

For example:

```
{{ value|urlencode }}
```

When value is "msg=Hello&World" then the output is "msg%3DHello%26World".

Forms

pickle

- Module: *mod_base* (page 287)

Pickle an Erlang value so that it can be safely submitted with a form.

The Erlang value is encoded using *erlang:term_to_binary/1* and signed with the site's secret key. In Erlang the value can be unpacked using *z_utils:depickle/2*

Usage:

```
<input type="hidden" name="confirm" value="{{ 'Hello world' | pickle }}" />
```

This will generate something like:

```
<input type="hidden" name="confirm" value="duZTXcXaxuruD3dhpt-
↪rxWokrhuDbQAAAAtIZWxsbyB3b3JsZA" />
```

HTML

show_media

- Module: *mod_base* (page 287)

Convert the image markers in HTML from the Rich Text editor into image tags.

When you add images in the Rich Text editor of the Zotonic admin, the HTML body text does not store `<img>` tags, but instead, special markers containing the picture id plus size and alignment hints.

The `show_media` tag converts these special markers (which are in fact HTML comments) back into image tags. For this, it uses the template called `_body_media.tpl`. This template is located in `mod_base` and can be overruled with your own.

Images that are inlined in the body text can have these parameters:

`alignment` Either `between`, `left` or `right`.

`size` Choose between `small`, `middle`, `large`. The actual sizes that are taken come from the config key `site.media_dimensions`, which defaults to `200x200,300x300,500x500`.

`crop` Checkbox if you want to crop your image to exactly the small/middle/large size. Otherwise it scales to fit in the bounding box of the respective size.

`link` Checkbox if you want to let the image include a link to its own page.

`caption` Caption that will be displayed below the media item.

These parameters can be set in the editor dialog 'Insert a Zotonic media item'.

See also:

embedded_media (page 382), *without_embedded_media* (page 383)

striptags

- Module: *mod_base* (page 287)

Removes all HTML tags from the value.

Useful as part of filtering input from external sources.

For example:

```
{{ value|striptags }}
```

When value is "Helloworld" then the output will be "Helloworld".

truncate_html

- Module: *mod_base* (page 287)

Truncate a HTML text to a maximum length.

The HTML text is truncated to the maximum length specified with the argument. The text will be truncated at the maximum length, if any text remains then the ellipsis character ... is added and all open HTML tags are closed.

For example:

```
{{ value|truncate_html:8 }}
```

If the value is `hello <b>world</b>` then the output is `hello <b>wo...</b>`.

Entities like "&" are counted as a single character.

Self closing entities like `<img/>` and `<br/>` are not counted as characters.

Truncating character

An optional second argument defines which text will be added if the text is truncated:

```
{{ value|truncate_html:8:" (more)" }}
```

If the value is `hello <b>world</b>` then the output is `hello <b>wo (more)</b>`.

See also:

truncate (page 405)

urlize

- Module: *mod_base* (page 287)

Find urls in the given input and make them clickable.

Example:

```
{{ "http://foo.bar/"|urlize }}
```

Outputs:

```
<a href="http://foo.bar/">http://foo.bar/</a>
```

This filter is very similar to the *escape_link* (page 378) filter.

See also:

escape_link (page 378)

embedded_media

- Module: *mod_base* (page 287)

Fetch media ids that are embedded in the `body`, `body_extra` and *text* blocks of your page.

This filter lets you loop over every image that is embedded in the texts of the given page:

```
{% for media_id in id|embedded_media %}
    {% media media_id width=315 extent %}
{% endfor %}
```

There is an optional (boolean) argument to only fetch media ids from the `body` and `body_extra` properties:

```
{% for media_id in id|embedded_media:0 %}
    {% media media_id width=315 extent %}
{% endfor %}
```

You can also fetch all media ids embedded in a text:

```
{% for media_id in id.body|embedded_media %}
    {% media media_id width=315 extent %}
{% endfor %}
```

See also:

show_media (page 381), *without_embedded_media* (page 383)

without_embedded_media

- Module: *mod_base* (page 287)

Filter out media ids that are embedded in the `body`, `body_extra` and *text* blocks of your page.

This filter lets you loop over every image that is not included in the embedded `body` texts of the given page. This makes it easy to only show images that have not been shown already:

```
{% for media_id in m.rsc[id].media|without_embedded_media:id %}
    {% media media_id width=315 extent %}
{% endfor %}
```

The only argument to the filter is the *id* of the page that you want to consider for filtering from the body text.

There is an optional second argument to only consider media ids in the `body` and `body_extra` properties:

```
{% for media_id in m.rsc[id].media|without_embedded_media:id:0 %}
    {% media media_id width=315 extent %}
{% endfor %}
```

See also:

show_media (page 381), *embedded_media* (page 382)

Lists

after

- Module: *mod_base* (page 287)

Return the first element after another element in a list. For example:

```
{{ [1,2,3]|after:2 }}
```

Evaluates to the value `3`.

If the element is not part of the list, or is the last element in the list, the returned value is `undefined`.

before

- Module: *mod_base* (page 287)

Return the first element before another element in a list. For example:

```
{{ [1,2,3]|before:2 }}
```

Evaluates to the value `1`.

When the element is not part of the list, or is the first element in the list, the returned value is `undefined`.

chunk

- Module: *mod_base* (page 287)

This filter splits a list in shorter lists. It splits an array in sub-arrays of at most a given length. This is useful when displaying a list of items in columns or rows.

For example:

```
{% for s in [1,2,3,4,5]|chunk:2 %}
    {% for n in s %}{{ n|format_number }} {% endfor %} *
{% endfor %}
```

This displays `1 2 * 3 4 * 5 *`, as the array is split in three chunks. The last chunk is not filled to the maximum length. Then number of chunks depends on the length of the input list, this in contrary to the split_in filters where the number of splits is fixed and the length per split is variable.

See also:

split_in (page 390), *vsplit_in* (page 391)

exclude

- Module: *mod_base* (page 287)

Filters a list on the value of a property, either on absence or inequality.

This is the inverse of *filter* (page 385).

Testing presence

To filter a list of values:

```
{% print somelist|exclude:`p` %}
```

Results in a list where all elements **do not have** the property `p` defined and where the property (after conversion to boolean) is `false`.

This can be used to filter a list of resource ids on the absence of a property. For example, to see all unpublished elements in a list of resource ids:

```
{% print [1,2,3,4,5,6]|exclude:`is_published` %}
```

To find all pages from page connection `hasdocument` that **do not have** an image:

```
{% print id.o.hasdocument|exclude:`depiction` %}
```

Testing equality

A second argument can be added to test on inequality:

```
{% print somelist|exclude:`title`:"Untitled" %}
```

Shows all elements whose `title` property **is not** "Untitled".

Below is another example of inversely filtering a list:

```
{% with m.search[{latest cat='gallery'}] as result %}
  {% if result.total > 0 %}
    {%  with result|exclude:`name`:"page_home_gallery"|random as gallery_rsc_id %}
      {% include "_gallery_widget.tpl" id=gallery_rsc_id %}
    {% endwith %}
  {% endif %}
{% endwith %}
```

The example above filters against a search result and returns only elements whose `name` **is not** "page_home_gallery".

See also:

is_visible (page 399), *is_a* (page 398), *filter* (page 385)

filter

- Module: *mod_base* (page 287)

Filters a list on the value of a property, either on presence or equality.

Testing presence

To filter a list of values:

```
{% print somelist|filter:`p` %}
```

Results in a list where all elements have the property `p` defined and where the property (after conversion to boolean) is `true`.

This can be used to filter a list of resource ids on the presence of a property. For example, to see all published elements in a list of resource ids:

```
{% print [1,2,3,4,5,6]|filter:`is_published` %}
```

To find all pages from page connection `hasdocument` that have an image:

```
{% print id.o.hasdocument|filter:`depiction` %}
```

Testing equality

A second argument can be added to test on equality:

```
{% print somelist|filter:`title`:"Untitled" %}
```

Shows all elements whose `title` property is "Untitled".

See also:

is_visible (page 399), *is_a* (page 398), *exclude* (page 384)

group_by

- Module: *mod_base* (page 287)

Groups items of a list by a property.

When the item is an integer then it is assumed to be the id of a resource. This is especially useful for grouping items in for-loops.

For example:

```
{% for grp in value|group_by:"a" %} ... loop over grp ... {% endfor %}
```

When value is the three element list:

```
[
 [{a,1}, {b,1}],
 [{a,1},{b,2}],
 [{a,2},{b,3}]
]
```

then the output of group_by "a" will be the two element list:

```
[
 [[{a,1},{b,1}],[{a,1},{b,2}]],
 [[{a,2},{b,3}]]
].
```

index_of

- Module: *mod_base* (page 287)

Returns the index of the first occurrence of the item in the given list.

For example:

```
{{ [44,11,2,443,2]|index_of:11 }}
```

Returns `2`.

Note: Erlang list indices are always 1-based.

See also:

element (page 408)

is_list

- Module: *mod_base* (page 287)

Test if a value is a list:

```
{% if [1,2,3]|is_list %}Yes, this is a list {% endif %}
```

join

- Module: *mod_base* (page 287)

Joins the elements of a list. Joins the elements of the input list together, separated by the argument.

For example:

```
{{ value|join:", " }}
```

When value is the list `["hello", "world"]` then the output will be `"hello, world"`.

See also:

element (page 408), *tail* (page 390), *split* (page 390)

make_list

- Module: *mod_base* (page 287)

Forces the value to a list.

This is especially useful for loops using the `{% for %}` tag.

For example:

```
{{ value|make_list }}
```

When value is the tuple `{"a","b"}` then the output is the list `["a","b"]`.

When the input is a *model*, this filter forces the model to call its `m_to_list/2` function, which is used to iterate over an entire model, e.g.:

```
{% print m.req|make_list %}
```

This will show something like:

```
[{version,{1,1}},
 {peer,"127.0.0.1"},
```

Instead of printing just `{m, req, undefined}` when the filter was not applied.

member

- Module: *mod_base* (page 287)

Finds a value in a list.

Checks if the value is part of the argument. The argument must be a list. Returns a boolean value.

For example:

```
{% if value|member:[1,2,3] %}
    One of the first three
{% endif %}
```

When value is the integer 2 then the output is “One of the first three”.

nthtail

- Module: *mod_base* (page 287)

Fetch the nth tail of a list.

Useful when you want to skip the first N elements of a list when looping.

For example:

```
{% for a in value|nthtail:2 %}{{ a|format_number }}{% endfor %}
```

When value is the list `[1,2,3]` then the output is `3`.

See also:

first (page 366), *tail* (page 390)

random

- Module: *mod_base* (page 287)

Returns a random value from a list of values. When the input is an empty list or not a list then the result is undefined.

For example:

```
{{ ["a","b","c"]|random }}
```

The output of this is one of “a”, “b” or “c”.

See also:

randomize (page 388), *rand* (page 395)

randomize

- Module: *mod_base* (page 287)

Shuffle a list of values.

For example:

```
{{ ["a","b","c"]|randomize }}
```

The output of this is the same list, but the order of the elements randomized. So for instance: [”c”, “a”, “b”].

See also:

rand (page 395), *random* (page 388)

range

- Module: *mod_base* (page 287)

Generate a list of integers, with an optional step.

For example:

```
{{ 1|range:4 }}
```

Generates the list `[1,2,3,4]`.

The second filter argument is the step size:

```
{{ 1|range:10:2 }}
```

Generates the list `[1,3,5,7,9]`.

reversed

- Module: *mod_base* (page 287)

Reverse a list.

For example:

```
{{ value|reversed }}
```

When value is [”hello”, “world”] then the output is “worldhello”.

The main use for this filter is to reverse lists of values or search results. There is no support for multi-byte unicode characters, this is only a problem when applying the filter directly to a string value.

New in version 0.6.

slice

- Module: *mod_base* (page 287)

Perform array-slice operations on a list.

Given a list = `[1,2,3,4,5,6,7,8,9,0]`

Get all elements from element M to element N:

```
{{ list|slice:[3,7] }} -> [3,4,5,6,7]
{{ list|slice:[3,-3] }} -> [3,4,5,6,7]
{{ list|slice:[-7,-3] }} -> [3,4,5,6,7]
{{ list|slice:[-7,7] }} -> [3,4,5,6,7]
```

Get all elements except the first N:

```
{{ list|slice:[3,] }} -> [3,4,5,6,7,8,9,0]
{{ list|slice:[-7,] }} -> [3,4,5,6,7,8,9,0]
```

Get all elements up to element N:

```
{{ list|slice:[,3] }} -> [1,2,3]
{{ list|slice:[3] }} -> [1,2,3]
```

Get all elements except the last N:

```
{{ list|slice:[,-3] }} -> [1,2,3,4,5,6,7]
{{ list|slice:[-3] }} -> [1,2,3,4,5,6,7]

{{ list|slice:[M,N] }}, where N < M will return []
{{ list|slice:[,] }}, where N < M will return [1,2,3,4,5,6,7,8,9,0]
```

sort

- Module: *mod_base* (page 287)

The *sort* filter takes a list of items to sort. Items can be a ordinary list of terms, or a list of resources to be filtered based on their properties. Sort order and properties to sort on are given as arguments to the filter.

By default it sorts the list in *ascending* order, and resource lists are sorted on their *id* if no property is specified.

Example:

```
{{ [4, 6, 2, 3, 5]|sort }}
```

Sorts the list of numbers in *ascending* order.

Example:

```
{{ [4, 6, 2, 3, 5]|sort:'desc' }}
```

Sorts the list of numbers in *descending* order.

Example:

```
{% for r in id.o.author|sort:['title', 'desc', 'modified'] %}
   do something with `r`...
{% endfor %}
```

This will sort on *title* in *ascending* order first, then on *modified* in *descending* order. Any number of properties may be added, each one can have it's own sort order, or use the current one.

See *m_rsc* (page 426) for a list of properties available to sort on.

Sort order may be either *ascending* or *descending* (may be abbreviated as *asc*, +, *desc*, - or as string version of those).

split

- Module: *mod_base* (page 287)

Splits the filter value into a list of values.

The input value is split by the filter argument, for example:

```
{{ "foo bar baz"|split:" " }}
```

Will create the list `["foo", "bar", "baz"]`.

See also:

join (page 386)

split_in

- Module: *mod_base* (page 287)

This filter split a list in shorter lists. It splits an array in N sub-arrays of more or less equal length. This is useful when displaying a list of items in columns.

For example:

```
{% with [1,2,3,4,5,6]|split_in:3 as a,b,c %}
    {% for n in a %}{{ n|format_number }} {% endfor %}
{% endwith %}
```

This displays `1 4`. The variable b will be `[2,5]` and the variable c will be `[3,6]`.

See also:

chunk (page 384), *vsplit_in* (page 391)

tail

- Module: *mod_base* (page 287)

Fetch the tail of a list.

Returns the tail of a list. Useful when you want to skip the first element of a list when looping.

For example:

```
{% for a in value|tail %}{{ a|format_number }}{% endfor %}
```

When value is the list `[1,2,3]` then the output is `23`.

See also:

first (page 366), *nthtail* (page 387)

vsplit_in

- Module: *mod_base* (page 287)

This filter splits a list in shorter lists. It splits an array in N sub-arrays of more or less equal length. This is useful when displaying a list of items in columns.

Note that it splits the array in a different way than *split_in* (page 390) does: The filter *split_in* takes alternating elements from the array, where *vsplit_in* takes complete runs at a time. See the example below.

For example:

```
{% with [1,2,3,4,5,6]|vsplit_in:3 as a,b,c %}
    {% for n in a %}{{ n|format_number }} {% endfor %}
{% endwith %}
```

This displays `1 2`. The variable b will be `[3,4]` and the variable c will be `[5,6]`.

See also:

chunk (page 384), *split_in* (page 390)

without

- Module: *mod_base* (page 287)

Remove the items given in the argument from the filter value.

For example:

```
{% print [1,2,3]|without:[2] %}
```

prints:

```
[1,3]
```

This filter also works on list-like values like resource edges:

```
{% for id in id.o.tags|without:some_id.o.tags %}
```

Iterates of all *tags* edges of the given id, for each id that is not also an edge of *some_id*.

Mailing list

inject_recipientdetails

- Module: *mod_mailinglist* (page 299)

Adds recipient query string details to hyperlinks.

This filter is meant for use inside e-mail templates. It replaces each occurrence of `##` with the details of the subscriber that the mail is sent to, encoded as query string arguments.

Each occurrence of `##` will be transformed to recipient details in the following form:

```
?email=foo@bar.com&name_first=John&name_last=Doe
```

Its use case is when sending a mailing with a link in it which arrives at a webpage where the user has to enter his e-mail address and name details. Using this filter, those parameters can be conveniently be pre-filled.

Menu

menu_flat

- Module: *mod_menu* (page 301)

Flattens the rsc menu structure for use in a template loop.

Example:

```
{% for item in m.rsc[id].menu|menu_flat %}
...
{% endif %}
```

See also:

menu_subtree (page 392), *menu_trail* (page 393)

menu_is_visible

- Module: *mod_menu* (page 301)

Filters a list of menu items on visibility. The *is_visible* (page 399) filter can't be used due to the structure of a menu item list.

Example:

```
{% with m.rsc.main_menu.menu|menu_is_visible as menu
        {% if menu %}
        <ul>
                {% for id,subids in menu %}
                        <li>{{ id.title }}</li>
                {% endfor %}
        </ul>
        {% endif %}
{% endwith %}
```

menu_rsc

- Module: *mod_menu* (page 301)

Todo

Not yet documented.

menu_subtree

- Module: *mod_menu* (page 301)

Get the subtree of an id in a menu (if any).

Returns the subtree of the filter value. Useful for showing a part of a menu when browsing sub-pages.

If the given id is not found inside the menu, it returns `undefined`.

If no argument is given, it takes menu from the resource with the name `main_menu`.

See also:

menu_trail (page 393), *menu_flat* (page 392)

menu_trail

- Module: *mod_menu* (page 301)

Return a breadcrumb navigation trail for the given id.

This filter locates the filter value which represents the current page in the main menu or in the menu saved in the resource of the given id.

For example:

```
{% print id|menu_trail:55 %}
```

Could print the list `[13, 33]` if ids 13 and 33 are the parents of the *id* argument in the menu resource 55.

If no argument is given, it takes menu from the resource with the name `main_menu`.

Showing Menu Trail only for submenu items

It is sometimes useful to suppress the menu trail on top level items. Here is how to do it.

The *menu_trail* (page 393) includes the whole path through the menu to the current page if it is reachable that way. Sometimes it may seem pointless to show the menu trail if we are on the first level of the menu. If we want to avoid this we need to avoid rendering a trail when it is less than two items long.

One simple condition change to *_article_chapeau.tpl* from the *blog* skeleton makes this work:

```
{% with id|menu_trail as parents %}
  {% if parents|length > 1 %}
  <h5 class="chapeau">
  {% for p in parents %}
  <a href="{{ m.rsc[p].page_url }}">{{ m.rsc[p].title }}</a>
  {% if not forloop.last %}&raquo;{% endif %}
{% endfor %}</h5>{% endif %}{% endwith %}
```

The key here is `{% if parents|length > 1 %}` in place of just `{% if parents %}`.

The *if* (page 447) tag is now rendering the menu_trail only if there are two or more items in it which - as I mentioned before - happens when you are at least two levels deep in the menu.

See also:

menu_subtree (page 392), *menu_flat* (page 392)

Miscellaneous

gravatar_code

- Module: *mod_comment* (page 287)

Calculate the gravatar code for an e-mail address:

```
{{ "arjan@scherpenisse.net"|gravatar_code }}
```

Will output:

```
3046ecab06c4f9cdb49963a96636e5ef
```

This hash can then be used for displaying Gravatar images.

twitter

- Module: *mod_twitter* (page 318)

Format a plain text Tweet into HTML.

This filter creates hyperlinks out of the embedded URLs, @-references and hashtags in a Tweet, like Twitter does.

For example:

```
{{ "@acscherp"|twitter }}
```

Converts into:

```
<a href="http://twitter.com/acscherp">@acscherp</a>
```

Numbers

filesizeformat

- Module: *mod_base* (page 287)

This filter formats a numeric value as KB, MB etc. This filter can be used to display a number of bytes in a human readable format of kilo- or megabytes.

For example:

```
{{ 532671|filesizeformat }}
```

Will output `520.2 KB`.

is_even

- Module: *mod_base* (page 287)

Test if an integer value is even:

```
{% if 2|is_even %}Yes, the number 2 is even{% endif %}
```

is_number

- Module: *mod_base* (page 287)

Test if a value is a number (integer or floating point):

```
{% if 1|is_number %}Yes, this is a number{% endif %}
```

max

- Module: *mod_base* (page 287)

Take the maximum of the filter value and its first argument.

The following:

```
{% print 102|to_integer|max:103 %}
```

Prints `103`.

See also:

min (page 395), *minmax* (page 395)

min

- Module: *mod_base* (page 287)

Take the minimum of the filter value and its first argument.

The following:

```
{% print 102|to_integer|min:103 %}
```

Prints `102`.

See also:

max (page 394)

minmax

- Module: *mod_base* (page 287)

Force the given value in the given range.

This clamps the filter value between the two filter arguments.

Example:

```
{% print 3|to_integer|minmax:10:20 %}
```

This will print `10`, since that is the minimum value allowed.

Passing in `undefined` will not clamp the value but return `undefined`.

See also:

max (page 394), *min* (page 395)

rand

- Module: *mod_base* (page 287)

Generates a random number. The number is from, and including, 1 up to, and including, the input value.

Example:

```
{{ 100|rand|format_integer }}
```

Might output "42".

The rand filter can also generate a floating point value by given it a floating point number or a string representing a floating point number as input. It will generate a number from, but not including, 0 to, and including, the input value:

```
{{ "4.0"|rand|format_number }}
```

Might output "3.1415926536"

See also:

randomize (page 388), *random* (page 388)

to_integer

- Module: *mod_base* (page 287)

Convert the input to an integer value.

Example:

```
{{ "123"|to_integer }}
```

Results in the integer value `123`.

This filter uses the `z_convert:to_integer/1` function.

See also:

to_binary (page 367), *format_number* (page 402), *format_integer* (page 402)

Regular Expressions

match

- Module: *mod_base* (page 287)

Match a value with a regular expression.

Returns true if the value matches the regular expression. This is handy for checking if a string starts or ends with a particular value.

Usage:

```
{% if value|match:".*foo$" %}
```

Checks if the value ends with "foo".

replace

- Module: *mod_base* (page 287)

Regular expression replacement of a pattern with a string.

Replaces the sub strings matching a regular expression with a new string value.

For example:

```
{{ "abcba"|replace:["b","x"] }}
```

The output will be the string "axcxa".

If you do not specify the replacement value, the replacement value is assumed to be the empty string:

```
{{ "abcba"|replace:"b" }}
```

The output will be the string "aca".

Note: This filter is inefficient, as it will compile and match a regular expression while serving the template. Try to do string replacements when you *save* your content, and not when you serve the content.

Resource lists

group_firstchar

- Module: *mod_base* (page 287)

Group a list of *sorted resource* (page 426) ids on their first letter of the title or another rsc property. After grouping, it splits this list in a number of more-or-less even columns.

This is useful for displaying multiple columns of alphabetically sorted pages, in which the pages are grouped by the first letter, for instance a search result.

For instance, this piece of template code:

```
<table>
{% for cols in m.search[{query cat="country" sort="pivot_title"}]|group_firstchar:
↪"title":3 %}
   <td>
      {% for group in cols %}
      <b>{{ group.first }}</b>
      {% for id in group.result %}
      <li>
          {{ id.title }}
      </li>
      {% endfor %}
   {% endfor %}
   </td>
{% endfor %}
</table>
```

Groups the list of ids by title in three columns. It then uses nested for-loops to render a table similar to this:

```
+----------------+----------------+----------------+
|A               |D               |G               |
| Afghanistan    | Denmark        | Georgia        |
| Azerbeidjan    |                | Germany        |
|                |E               | Grenada        |
|B               | Ecuador        | Guadeloupe     |
| Belgium        | Egypt          | Greenland      |
| Belarus        | Eritrea        | Guinea         |
| Brazil         | Estonia        |                |
|                |                |H               |
|C               |F               | Haiti          |
| Canada         | Fiji           |                |
|                | Finland        |                |
+----------------+----------------+----------------+
```

As you can see, all three columns have approximately the same size, although the size of the indiviual groups differs.

When no nr. of columns is given, the groups are returned in a single column.

See also:

group_title_firstchar (page 397)

group_title_firstchar

- Module: *mod_base* (page 287)

Similar to *group_firstchar* (page 397), but always uses the `title` column from the rsc table.

This is merely a shortcut, simplifying the template syntax:

```
<table>
{% for cols in m.search[...]|group_title_firstchar:4 %}
   <td>
      {% for group in cols %}
      <b>{{ group.first }}</b>
      {% for id in group.result %}
      <li>
          {{ id.title }}
      </li>
      {% endfor %}
   {% endfor %}
   </td>
{% endfor %}
</table>
```

Groups alphabetically on title, in four columns.

See also:

group_firstchar (page 397)

is_a

- Module: *mod_base* (page 287)

Filter a list of resource ids on category, or test if a single resource id belongs to a category.

This filter can be applied to a list of resource ids or a single resource id.

When it is applied to a list then it will filter the list of ids. Only those resource ids that belong to a certain category remain. Optionally the filter only returns the first n matches.

When applied to a single integer (resource id), then it will return a boolean. True when the id belongs to the parameter's category, false otherwise.

Apply to a single resource id

Example:

```
{{ 1|is_a:"person"|yesno }}
```

Will output "yes", because the resource with id 1 is a person (the System Administrator).

Apply to a list of resource ids

When applied to a list of ids:

```
{% for part_id in m.rsc[id].o.haspart|is_a:"person" %}
    {{ m.rsc[part_id].title }}
{% endfor %}
```

This will list all collection members that are a person. While:

```
{% for part_id in m.rsc[id].o.haspart|is_a:"person":3 %}
    {{ m.rsc[part_id].title }}
{% endfor %}
```

Lists only the first three collection members that are a person.

See also:

is_not_a (page 399), *is_visible* (page 399), *filter* (page 385)

is_not_a

- Module: *mod_base* (page 287)

is_not_a mirrors *is_a* (page 398). It is particularly useful when iterating over a category and excluding members of a sub-category (iterating over all images associated with a page except images in the thumbnail category).

Example for looping over all media in a rsc but excluding the thumbnail resources:

```
{% for m in m.rsc[id].media|is_not_a:"thumbnail" %}
...
{% endfor %}
```

See also:

is_a (page 398), *is_visible* (page 399), *filter* (page 385)

is_visible

- Module: *mod_base* (page 287)

Filter a list of resource ids so that only the visible ids remain.

This filter can be applied to a list of resource ids or a single resource id.

This filter can be applied to a list of resource ids. Only those resource ids that are visible for the current user remain. Optionally the filter only returns the first n matches.

An example:

```
<ul>
{% for part_id in m.rsc[id].o.haspart|is_visible %}
    <li>{{ m.rsc[part_id].title }}</li>
{% endfor %}
</ul>
```

This will list all collection members that are visible, preventing empty list items.

Whilst:

```
{% for part_id in m.rsc[id].o.haspart|is_visible:3 %}
    {{ m.rsc[part_id].title }}
{% endfor %}
```

Lists only the first three collection members that are visible.

See also:

is_a (page 398), *is_not_a* (page 399), *filter* (page 385)

Resources

admin_merge_diff

- Module: *mod_admin_merge* (page 283)

Todo

Not yet documented.

content_type_label

- Module: *mod_base* (page 287)

Todo

Not yet documented.

content_type_urls

- Module: *mod_base* (page 287)

Todo

Not yet documented.

summary

- Module: *mod_base* (page 287)

Extract a summary from a resource.

The value of the filter is the resource's id.

This uses the *summary* field if available, but if it is empty, uses the first paragraph of the body text to show a summary.

Useful for displaying excerpts of texts that do not always have a separate summary provided.

Example:

```
{{ id|summary }}
```

temporary_rsc

- Module: *mod_admin* (page 275)

Creates a temporary resource if its input value is not defined.

The created resource receives the properties of the second parameter. The resource is tied to the lifetime of the session, if the session stops then the resource is deleted if it has not been changed in the meantime.

Only a single temporary resource can exist per category per session. Requesting a second resource of the same category for a session will return the earlier created resource. If the earlier resource has been updated, then a new resource is created.

This filter is used for situations where an edit form is needed but an intermediate dialog requesting for the title of the resource is unwanted.

Example:

```
{% with id|temporary_rsc:{props title=_"New" category=`article`} as id %}
    {% wire id=#form type="submit" postback=`rscform` delegate=`controller_admin_
↪edit` %}
    <form id="{{ #form }}" action="postback">
        <input type="hidden" name="id" value="{{ id }}" />
        <input type="hidden" name="is_published" value="1" />
        <input type="text" name="title" value="{{ id.title }}" />
        ...
    </form>
{% endwith %}
```

See *mod_admin* (page 275)

Strings

append

- Module: *mod_base* (page 287)

Appends the argument to the value.

For example:

```
{{ value|append:" world" }}
```

When value is `hello` then the output will be `hello world`.

See also:

insert (page 403)

capfirst

- Module: *mod_base* (page 287)

Converts the first character of the value to uppercase.

For example:

```
{{ value|capfirst }}
```

When value is `hello world` then the output is `Hello world`.

At the moment this only works for the characters a through z. Accented characters (like ü) are not yet supported.

See also:

upper (page 406)

center

- Module: *mod_base* (page 287)

Centers the value in a field of a certain width using spaces.

For example:

```
{{ value|center:7 }}
```

When value is `hello` then the output is _`hello`_ (with spaces).

Centering only works for single byte character values. At this moment there is no support for multi-byte unicode characters.

See also:

rjust (page 404), *ljust* (page 403)

format_integer

- Module: *mod_base* (page 287)

Show an integer value.

Formats an integer value as a list, assuming a radix of ten.

For example:

```
{{ value|format_integer }}
```

When the value is the integer 123 then the output is the list `123`.

The format_integer filter has an optional argument to pad the returned string with zeros to a fixed length. For example:

```
{{ 123|format_integer:5 }}
```

Will output `00123`. And when the number does not fit:

```
{{ 123|format_integer:2 }}
```

Will output "**".

Note: This option only works for positive integers.

See also:

to_integer (page 396), *format_number* (page 402), *format_price* (page 402)

format_number

- Module: *mod_base* (page 287)

Show an integer or float.

Formats integer and float values as a list, assuming a radix of ten.

For example:

```
{{ value|format_number }}
```

When the value is the float `12.0` then the output is the list `12.0`.

See also:

format_integer (page 402), *format_price* (page 402)

format_price

- Module: *mod_base* (page 287)

Show a price with decimals.

Formats integer and float values as a number with two decimals.

For example:

```
{{ value|format_price }}
```

When the value is the float `12.1` then the output is the list `12.10`.

An undefined price will have the output "-".

Note: the decimal separator is currently always a dot, independent of the user's language.

See also:

format_number (page 402), *format_integer* (page 402)

insert

- Module: *mod_base* (page 287)

Prepends the argument in front of the value.

For example:

```
{{ value|insert:"world " }}
```

When value is "hello" then the output will be "world hello".

See also:

append (page 401)

is_valid_email

- Module: *mod_email_status* (page 293)

Todo

Not yet documented.

ljust

- Module: *mod_base* (page 287)

Justifies the value in a field of a certain width to the left, with spaces.

For example:

```
{{ value|ljust:7 }}
```

When value is `hello` then the output is `hello__` (with spaces).

Justifying only works for single byte character values. At this moment there is no support for multi-byte unicode characters.

See also:

rjust (page 404), *center* (page 401)

lower

- Module: *mod_base* (page 287)

Translates the value to lower case.

For example:

```
{{ value|lower }}
```

When value is "Hello World" then the output is "hello world".

Note: There is partial support for multi-byte unicode characters.

See also:

upper (page 406)

replace_args

- Module: *mod_base* (page 287)

Replace `$N` placeholders in string from a list of replacement values.

The input string is searched for any `$N` tags, which is replaced with the *N*'th value from the list of arguments supplied to the filter.

Example usage:

```
{{ "Replace $1 and give it some $2."|replace_args:["item", "meaning"] }}
```

Will result in: “Replace item and give it some meaning.”

The `$N` tag may be escaped by \ to avoid replacement.

N may be in the range [1..9]. If *N* is out of range of the provided args, the *$N* tag is left as-is.

The *$N* tags may come in any order, any number of times.

When replacing a single tag, the replacement argument may be given directly:

```
{{ "John is $1"|replace_args:"the one" }}
```

Outputs: “John is the one”

New in version 0.8.

rjust

- Module: *mod_base* (page 287)

Justifies the value in a field of a certain width to the right, using spaces.

For example:

```
{{ value|rjust:7 }}
```

When value is `hello` then the output is `__hello` (with spaces).

Justifying only works for single byte character values. At this moment there is no support for multi-byte unicode characters.

See also:

ljust (page 403), *center* (page 401)

stringify

- Module: *mod_base* (page 287)

Translates atoms, integers and floats to strings. The undefined value is translated to the empty string. Does not translate tuples.

For example:

```
{{ value|stringify }}
```

When *value* is undefined then the output will be “”.

See also:

slugify (page 380), *to_binary* (page 367)

tokens

- Module: *mod_base* (page 287)

Returns a list of tokens from input string, separated by the characters in the filter argument.

This is filter maps directly onto the `string:tokens/2` function in `stdlib` from the Erlang/OTP distribution.

Example:

```
{{ "abc defxxghix jkl"|tokens:"x " }}
```

Will give the list of tokens: `["abc", "def", "ghi", "jkl"]`.

trim

- Module: *mod_base* (page 287)

Removes whitespace at the start and end of a string.

For example:

```
{{ value|trim }}
```

When the value is `" hello "` then the output is `"hello"`.

Internally, this calls `z_string:trim/1` to perform the trimming.

truncate

- Module: *mod_base* (page 287)

Truncate a text to a maximum length.

The text is truncated to the maximum length specified with the argument. The text is always truncated at a word boundary. If the truncation is not after punctuation then the unicode ellipsis … character is appended.

For example:

```
{{ value|truncate:8 }}
```

If the value is `hello world.` then the output is `hello...`.

Entities like `&` are counted as a single character.

This filter is multibyte aware: Multi-byte UTF-8 characters are assumed to be non-breaking characters.

Truncating character

An optional second argument defines which text will be added if the text is truncated:

```
{{ value|truncate:8:" (more)" }}
```

If the value is `hello world.` then the output is `hello (more)`.

See also:

truncate_html (page 381)

upper

- Module: *mod_base* (page 287)

Translates the value to upper case.

For example:

```
{{ value|upper }}
```

When value is “Hello World” then the output is “HELLO WORLD”.

Note: There is partial support for multi-byte unicode characters.

See also:

capfirst (page 401), *lower* (page 403)

Survey

survey_answer_split

- Module: *mod_survey* (page 316)

Todo

Not yet documented.

survey_as_pages

- Module: *mod_survey* (page 316)

Split the page blocks into pages, prepare them for easy display in the survey question editor.

See *mod_survey* (page 316)

Todo

Not yet documented.

survey_is_stop

- Module: *mod_survey* (page 316)

Check if there is a 'stop' question in list of (survey) blocks

See *mod_survey* (page 316)

Todo

Not yet documented.

survey_is_submit

- Module: *mod_survey* (page 316)

Todo

Not yet documented.

survey_prepare_matching

- Module: *mod_survey* (page 316)

Todo

Not yet documented.

survey_prepare_narrative

- Module: *mod_survey* (page 316)

Todo

Not yet documented.

survey_prepare_thurstone

- Module: *mod_survey* (page 316)

Todo

Not yet documented.

Translation

is_rtl

- Module: *mod_translation* (page 317)

Check if the given language is a rtl or ltr language.

Example:

```
{% if z_language|is_rtl %}
    You are browsing in an RTL language
{% endif %}
```

It currently returns `true` only for Arabic (`ar`), Farsi (`fa`) and Hebrew (`he`).

language

- Module: *mod_translation* (page 317)

Todo

Not yet documented.

trans_filter_filled

- Module: *mod_base* (page 287)

Filters all empty translations from a property.

This is used if it is important to show a text, but not all translations are filled in.

The filter takes as input a resource or other variable and as argumnent the property to be shown.

Example usage:

```
{{ id|trans_filter_filled:`body` }}
```

If the resource `id` has the body property:

```
{trans, [{en, <<>>}, {nl,<<"Hallo">>}]}
```

Then this will show `Hallo`, even if the language is set to `en`.

Tuples

element

- Module: *mod_base* (page 287)

Select an element from a tuple or list of tuples.

For example:

```
{{ value|element:1 }}
```

When value is a list of tuples `[{312,0.34}, {200,0.81}]` then the output is the list `[312,200]`.

When value is just a tuple, `{123, 22, 11}`, the output of `|element:1` is `123`.

See also:

before (page 383), *after* (page 383)

Variables

as_atom

- Module: *mod_base* (page 287)

Convert a value to an Erlang atom.

Atoms are represented in the template language as strings between backticks, `` `like this` ``. Some template function require their arguments to be atoms, and this filter helps in converting any value to such an atom:

```
{{ "foobar"|as_atom }}
```

evaluates to the atom `foobar`.

default

- Module: *mod_base* (page 287)

Provide an alternative value in case a value has a falsy value (0, `false`, `undefined` or empty string).

For example:

```
{{ value|default:1 }}
```

See also:

if (page 409), *is_defined* (page 410), *is_undefined* (page 410), *if_undefined* (page 409)

if

- Module: *mod_base* (page 287)

Selects an argument depending on a condition.

For example:

```
{{ value|if:"yes":"no" }}
```

This is a shortcut for using the *if* (page 447) tag. The same can be expressed as follows:

```
{% if value %}yes{% else %}no{% endif %}
```

Note that falsy values (0, `false`, `undefined` or empty string) evaluate to false.

Elaborate examples

```
{% with is_i18n|if:r.translation[lang_code].body:r.body as body %}
```

So if `is_i18n` evaluates to true, `body` is assigned to `r.translation[lang_code].body`, else to `r.body`.

```
{% include "_language_attrs.tpl" id=pid class=(pid==id)|if:"active":"" %}
```

Add parameter `class` to the included template; when `pid` equals `id`, `class` is `"active"`, otherwise an empty string.

See also:

if (page 447), *if_undefined* (page 409)

if_undefined

- Module: *mod_base* (page 287)

Tests whether a value is undefined, returning the given argument.

Whereas the *|default* filter also falls back to the default value when a value is an empty string or `false`, this filter *only* falls back to its value when the input value is the Erlang `undefined` atom.

This can be used for setting values which default to true if they are never set.

For example:

```
{% if value|if_undefined:`true` %}The value is true or undefined{% endif %}
```

If the value is `undefined`, the output will be "The value is true or undefined".

See also:

is_defined (page 410), *is_undefined* (page 410), *if* (page 409)

is_defined

- Module: *mod_base* (page 287)

Tests if a value is defined.

Checks if the value is not empty and outputs a boolean true or false. This is useful in combination with the *if* (page 447) tag.

For example:

```
{% if value|is_defined %}The value was defined{% endif %}
```

When the value is “foo” then the output “The value was defined”.

See also:

is_undefined (page 410), *if_undefined* (page 409), *if* (page 409)

is_undefined

- Module: *mod_base* (page 287)

Tests if a value is undefined.

Checks if the value is empty and outputs a boolean true or false. This is useful in combination with the *if* (page 447) tag.

For example:

```
{% if value|is_undefined %}The value was undefined{% endif %}
```

When the value is “” then the output “The value was undefined”.

See also:

is_defined (page 410), *if_undefined* (page 409), *if* (page 409)

make_value

- Module: *mod_base* (page 287)

Todo

Not yet documented.

---

pprint

- Module: *mod_base* (page 287)

Pretty print a zotonic value in a template.

Pretty printing a zotonic value in a template is handy during development. It outputs the value of an erlang variable in Html.

Usage:

```
{{ value | pprint }}
```

This output is similar to the *print* (page 454) tag, only are the values of the pprint filter not wrapped in `<pre>` tag.

See also:

print (page 454)

to_json

- Module: *mod_base* (page 287)

Display any value as in JSON (JavaScript Object Notation).

For example:

```
{{ [1,2,3]|to_json }}
```

Converts this list to a valid JSON UTF-8 encoded string which can be directly used in JavaScript calls.

The input value may be in a different encoding, given as argument to the *to_json* filter:

```
{{ value|to_json:"latin-1" }}
```

Currently, there are two encodings supported:

- UTF-8 (`utf-8`) i.e. no encoding conversion, which is the default when no argument is given.
- ISO 8859-1 (`latin-1`).

Models

See also:

Models (page 23)

m_acl

- Module: core

The m_acl model gives access the id of the currently logged in user, and provides a mechanism to do basic access control checks.

The following m_acl model properties are available in templates:

Property	Description
user	Returns the current user id. If not logged in, this returns `undefined`.
is_admin	Check if the current user is alllowed to access the admin. Internally, this checks the `use, mod_admin_config` ACL.
use, admin, view, delete, update, insert, link	These properties are shortcuts to check if the current user is allowed to do some action.
is_allowed	Perform custom ACL checks which are different from the ones mentioned.
authenticated	Used before the other ACL checks to check if a *typical* user is allowed to perform some actions. Example: `m.acl.authenticated.insert.article` If an user is logged on the that user's permissions are used.

This example prints a greeting to the currently logged in user, if logged in:

```
{% if m.acl.user %}
    Hello, {{ m.rsc[m.acl.user].title }}!
{% else %}
    Not logged in yet
{% endif %}
```

This example checks if the user can access the admin pages:

```
{% if m.acl.is_admin %} You are an admin {% endif %}
```

This example performs a custom check:

```
{% if m.acl.is_allowed.use.mod_admin_config %}
    User has rights to edit the admin config
{% endif %}
```

And to check if a resource is editable:

```
{% if m.acl.is_allowed.update[id] %}
   User can edit the resource with id {{ id }}
{% endif %}
```

A short hand for the above is (assuming *id* is an integer):

```
{% if id.is_editable %}
   User can edit the resource with id {{ id }}
{% endif %}
```

m_acl_rule

- Module: *mod_acl_user_groups* (page 271)

Not yet documented.

m_acl_user_group

- Module: *mod_acl_user_groups* (page 271)

Not yet documented.

m_admin_blocks

- Module: *mod_admin* (page 275)

Not yet documented.

m_admin_menu

- Module: *mod_admin* (page 275)

This model holds the admin menu, which is built up by calling each module to add items to the menu.

Extending the admin menu

By implementing the `admin_menu` *notification* (page 46) you can add your own menu items to the admin menu. The `admin_menu` notification is a fold which build up the menu, allowing each callback to add and remove menu items as they wish.

For example, this add a menu separator and an "edit homepage" button to the "content" submenu:

```
-include_lib("modules/mod_admin/include/admin_menu.hrl").

observe_admin_menu(admin_menu, Acc, _Context) ->
  [
   #menu_separator{
     parent=admin_content},
   #menu_item{
      id=admin_edit_homepage,
      parent=admin_content,
      label="Edit homepage",
      url={admin_edit_rsc, [{id, page_home}]}}
   |Acc].
```

The default submenu names are *admin_content*, *admin_structure*, *admin_modules*, *admin_auth* and *admin_system*, but you are free to add your own submenus.

m_admin_rsc

- Module: *mod_admin* (page 275)

This model exposes some meta-information of the *m_rsc* (page 426) model.

Currently the only thing it does is retrieve the count of the pivot queue, the queue which decides which resources need re-indexing.

To view the number of items in the pivot queue, do the following:

```
{% print m.admin_rsc.pivot_queue_count %}
```

m_backup

- Module: *mod_backup* (page 286)

Not yet documented.

m_backup_revision

- Module: *mod_backup* (page 286)

Not yet documented.

m_category

- Module: core

This model can retrieve information about the resource category hierarchy in different ways.

Categories are the principal categorization (typing) system of resources. Every page is assigned to exactly one category. Categories themselves are organized in a tree-like hierarchy.

A category is a resource with a special `category` record attached to it to store metadata related to the category hierarchy. The `m_category` model provides accessors to this category tree and individual category information.

An example of a category tree, as returned by `{% print m.category.tree %}`:

```
[[{id,101},
  {parent_id,undefined},
  {level,1},
  {children,[]},
  {path,"e"},
  {left,1000000},
  {right,1000000}],
 [{id,104},
  {parent_id,undefined},
  {level,1},
  {children,[[{id,106},
              {parent_id,104},
              {level,2},
              {children,[[{id,109},
                          {parent_id,106},
                          {level,3},
                          {children,[]},
                          {path,"hjm"},
                          {left,4000000},
                          {right,4000000}]]},
              {path,"hj"},
              {left,3000000},
              {right,4000000}]]},
```

```
  {path,"h"},
  {left,2000000},
  {right,4000000}],
  ...]
```

About the complete category tree

The following m_category model properties are available in templates:

Property	Description	Example value
tree	Return the complete forest of category trees as nested property lists.	See above.
tree2	Return the forest of category trees as nested property lists. Up to the children of the children.	See above.
tree_flat	Return a list of tuples for the category tree. This list is intended for select lists. There is a special field for the indentation. The returned list consists of proplists. The list does not contain the "meta" category, which contains the categories "predicate", "category" etc.	See above entries.
tree_flat_meta	Same as *tree_flat* but with the categories in the *meta* category.	See above entries.

About a single category

The m_category has some special properties defined when fetching a category, they are accessed by id or category name. For example:

```
{{ m.category[104].tree }}
{{ m.category.text.tree }}
```

Property	Description	Example value
tree	The category tree below and including the indexing category.	See above.
tree1	The list of direct children below the indexing category.	See above.
tree2	The category tree below and including the indexing category, up to the children of the children.	See above.
tree_flat	The category tree below and including the indexing category, up to the children of the children. As a flattened list.	See above.
path	List of parent category ids from the root till the category, excluding the indexing category.	[104, 106]
is_a	List of the parent category names form the root till the category, including the current category.	[text, article, news]
image	A random depiction for this category. The returned image filename comes from one of the pages within this category.	<<"2009/10/20/flat-world-proof.jpg">>
parent_id	The page id of the parent category. Returns an integer or, for a root category, undefined.	104
nr	The category nr. Used for building the tree, will change when categories are added or removed. An integer.	2
level	The depth of the category. Level 1 is the root, 2 and more are below the root.	1
left	The lowest value of the nr range of this category, including its sub categories.	2
right	The highest value of the nr range of this category, including its sub categories.	8
name	The unique page name of this category. A binary.	<<"text">>
path	The path through the hierarchy of categories to this category.	[104, 106]

m_comment

- Module: *mod_comment* (page 287)

Accesses comments on a page.

Comments are stored in the `comment` table. Comments are no separate rsc records because that will add many extra records and also because of access control restrictions.

When a page is not visible to a certain user then its comments shouldn't be visible as well. To simplify this check the comments are placed separate and made part of the rsc record.

This separate comment table also helps with cleaning up comments when the rsc record is deleted.

Todo

Finish m_comment

m_config

- Module: core

Zotonic has two places where a site's configuration is kept. One is in the site's config file (accessible through *m_site* (page 433)), the other in the `config` table. Entries in the config table overrule any module settings from the config file.

All m_config keys can be thought of as tuples `{Module, Key, Value}`, where Value is a complex value that can have a text value but also any other properties. Only configuration with text values can be edited in admin.

The config model table has different access methods. Below are examples of what is possible and how to access the configuration.

Fetch the value of a config key

Example, fetching the config key value of mod_emailer.from_email:

```
{{ m.config.mod_emailer.email_from.value }}
```

Where m.config.mod_emailer.email_from returns a property list which is much like:

```
[{id,5},
 {module,<<"mod_emailer">>},
 {key,<<"email_from">>},
 {value,<<"no-reply@example.com">>},
 {props,undefined},
 {created,{{2009,11,7},{20,48,27}}},
 {modified,{{2009,11,7},{20,48,27}}}
]
```

When the config key would have any extra value (besides the value property) then they would be visible as extra properties and the property "props" would not have been present.

When the configuration comes from the site config then the id property is not present.

Fetching all configurations of a module

This only returns configurations from the config table; configuration keys from the site config are not mixed in. This might change in the futurr:

```
{% for key, value in m.config.mod_emailer %}
    ...
{% endfor %}
```

Fetching all configurations

This only returns configurations from the config table. Configurations from the site config are not mixed in. This might change in the future:

```
{% for mod,keys in m.config %}
  {% for key,value in keys %}
    ...
  {% endfor %}
{% endfor %}
```

Listening for config changes

m_config uses z_notifier to broadcast events when values are changed or deleted:

```
#m_config_update{module=Module, key=Key, value=Value}
```

for "complex" property value updates/inserts a slighly different notification is used:

```
#m_config_update_prop{module=Module, key=Key, prop=Prop, value=Value}
```

For config key deletes, the `#m_config_update{}` record is broadcast with `undefined` for the value:

```
#m_config_update{module=Module, key=Key, value=undefined}
```

m_content_group

- Module: *mod_content_groups* (page 288)

Not yet documented.

m_custom_redirect

- Module: *mod_custom_redirect* (page 288)

Not yet documented.

m_edge

- Module: core

Access information about page connections.

Edges represent the connections between resources. They are implemented as tuples `{EdgeId, SubjectId, PredicateId, ObjectId, OrderNr}`. The edge id is a unique id representing the edge, it can be used with edit actions. The OrderNr defines the order of the edges *with respect to the subject*.

Most edge information is accessed using the *m_rsc* (page 426) model, but some information can only accessed with the m_edge model.

This model implements two template accessible options. They are mainly used to obtain the edge's id for edit pages.

The following m_edge model properties are available in templates:

Property	Description	Example value
o	Returns a function that accepts a page id and a predicate. The end result is a list of tuples {PageId, EdgeId} which are objects of the page. Example usage: `m.edge.o[id].author`	`[{204,13},{510,14},{508,15}]`
s	Identical to the "o" property, except that this function returns the subject edges.	
o_props	Similar to `m.edge.o[id].author` above, but returns a property list for the edges instead of the 2-tuple.	`[ {id, 86062}, {subject_id, 10635}, {predicate_id, 304}, {object_id, 57577}, {seq, 1}, {creator_id, 1}, {created, { {2015,11,17}, {11,23,32} }} ] ]`
s_props	Similar to `m.edge.s[id].author` above, but returns a property list for the edges instead of the 2-tuple.	
edges	Returns a function that accepts a page id. The end result is a list of edges per predicate where the predicate is an atom and the edges are property lists. Example usage: `m.edge[10635]`	See example below.
id	Look up an edge id by a subject/predicate/object triple. Example usage: `m.edge.id[subject_id].relation[object_id]` or: `m.edge.id[subject_id][predicate_name][object_id]` Returns `undefined` if the edge does not exist; otherwise returns an integer.	213

Example return value for `{% print m.edge[10635] %}`:

```
[{about,[[{id,86060},
         {subject_id,10635},
         {predicate_id,300},
         {name,<<"about">>},
         {object_id,17433},
         {seq,1},
         {created,{{2015,11,17},{11,22,11}}},
         {creator_id,1}]]},
 {author,[[{id,6},
```

```
            {subject_id,10635},
            {predicate_id,301},
            {name,<<"author">>},
            {object_id,10634},
            {seq,1000000},
            {created,{{2015,2,3},{16,23,20}}},
            {creator_id,1}]]}]
```

See also:

m_rsc (page 426), *m_media* (page 421)

m_email_receive_recipient

- Module: *mod_email_receive* (page 292)

Not yet documented.

m_email_status

- Module: *mod_email_status* (page 293)

Tracks the send/bounce/error status of all outgoing emails.

m_facebook

- Module: *mod_facebook* (page 293)

Not yet documented.

m_filestore

- Module: *mod_filestore* (page 294)

The filestore uses two tables for its administration.

Main administration table

The `filestore` table administrates which file is stored at what service. It also contains flags for flagging files that need to be deleted or moved from the remote service to the local file system.

The definition is as follows:

```
create table filestore (
    id serial not null,
    is_deleted boolean not null default false,
    is_move_to_local boolean not null default false,
    error character varying(32),
    path character varying(255) not null,
    service character varying(16) not null,
    location character varying(400) not null,
    size int not null default 0,
    modified timestamp with time zone not null default now(),
    created timestamp with time zone not null default now(),

    constraint filestore_pkey primary key (id),
    constraint filestore_path_key unique (path),
    constraint filestore_location_key unique (location)
)
```

The *path* is the local path, relative to the site's `files` directory. For example: `archive/2008/12/10/tycho.jpg`

The *service* describes what kind of service or protocol is used for the remote storage. Currently the service is always set to `s3`.

The *location* is the full url describing the location of the file on the remote service. For example: `https://s.greenqloud.com/mworrell-zotonic-test/archive/2008/12/10/tycho.jpg`

Upload queue

The upload queue holds file paths of newly added files that need to be uploaded. These file paths are relative to the `files` directory.

Periodically the system polls this table to see if any files need uploading. Any entry older than 10 minutes will be handled and an upload process will be started.

m_hierarchy

- Module: core

The category hierarchy tables have been replaced by *m_hierarchy*. This model defines named hierarchies of resources (pages).

If the categories are changed then the system needs to update the *pivot_category_nr* field of all resources. With the introduction of *m_hierarchy* this renumbering is much more efficient and will only affect a minimal number of resources.

The *m_hierarchy* module is also used for the content- and user group hierarchies, as used by the new *mod_acl_user_groups* module.

m_identity

- Module: core

The m_identity model manages usernames and other user identities. *mod_authentication* (page 286) uses it to store and check salted passwords, but also provides a safe storage for user tokens of any kind, as used by *mod_facebook* (page 293) and *mod_twitter* (page 318).

Note that a *user* does not have to be of the *person* category per se, in Zotonic anything can have identities attached to it.

The following m_identity model properties are available in templates:

Property	Description	Example value
is_user	Check if a page id is an user. Return a bool. Usage: m.identity[page_id].is_user	true
username	Fetch the username, if any, of an user. Returns a binary or undefined. Usage: m.identity[page_id].username	<<"admin">>

See also:

Access control (page 37), *mod_authentication* (page 286), *mod_twitter* (page 318) or *mod_facebook* (page 293).

m_import_csv_data

- Module: *mod_import_csv* (page 297)

Not yet documented.

m_l10n

- Module: *mod_l10n* (page 298)

Not yet documented.

m_log

- Module: *mod_logging* (page 298)

Not yet documented.

m_log_email

- Module: *mod_logging* (page 298)

Not yet documented.

m_mailinglist

- Module: *mod_mailinglist* (page 299)

Not yet documented.

m_media

- Module: core

Access to data about uploaded files and other media.

The `medium` (singular form of media) table stores all information of uploaded files or other media. Every resource can contain a single medium. A resource with a medium is most often of the category image, audio, video or document.

In template the `m_media` model is used to fetch the medium record by the resource id: `m.media[id]`. This is the same function as with `m.rsc[id].medium` except, that the m_rsc model does access control checks and the m_media does not.

The `m_media` model implements all functions to handle media files and is used by other Erlang modules.

Properties of a medium record

A medium record has minimally the following properties, other properties can be added by modules.

Property	Description	Example value
id	Id of the medium record, equal to the page id.	512
filename	Filename and path of the uploaded file, relative to the archive directory.	<<"2009/10/20/zotonic-datamodel.jpg">>
rootname	Root name of the filename.	<<"zotonic-datamodel">>
original_filename	Filename as suggested by the user agent when uploading the file. Can contain illegal characters.	<<"Zotonic-datamodel.jpg">>
mime	Mime type of the medium.	<<"image/jpeg">>
width	Width in pixels.	536
height	Height in pixels.	737
orientation	Exif oritentation of the image.	1
sha1	Optional sha1 checksum of uploaded file. Undefined when not present.	
size	Size in bytes of the uploaded file.	71585
preview_filename	Optional filename for a generated file preview.	
preview_width	Optional. Width of the generated preview.	
preview_height	Optional. Height of the generated preview.	
is_deletable_file	If the file should be deleted when the medium record is deleted. A boolean.	true
is_deletable_preview	If the optionally generated preview file should be deleted when the medium record is deleted. A boolean.	false
created	Timestamp when the medium record is created.	{{2009,10,20},{13,47,27}}

m_modules

- Module: core

Access information about which *modules* (page 39) are installed and which ones are active.

To test if a module is activated, for instance mod_signup:

```
{% if m.modules.active.mod_signup %}
    {# Do things that depend on mod_signup #}
{% endif %}
```

To print a list of all active modules:

```
{{ m.modules.all|pprint }}
```

m_notifier

- Module: core

Use the *notification system* (page 46) from your templates.

This model makes it possible to user the power of the notifier mechanism inside your templates.

m.notifier.first

Send the message between the square brackets to the notifier system. The value of the first reaction is used. The order in which the modules are called is based on the priority of the modules.

When rendered the templates sends a notification with the message `banner` to all listeners, rendering the first reaction:

```
{{ m.notifier.first[`banner`] }}
```

A module would implement the following callback:

```
observe_banner(banner, _Context) ->
    <<"<h1>This is a banner</h1>">>.
```

m.notifier.map

Send the message between the brackets to the notifier message. All reactions are returned in a list. The order in which the modules are called is based on the priority of the module.

Here is an example of the use of map to insert footer navigation items. This works as follows. When the template is rendered the zotonic notifier is called with the message `{footer_navigation, [{current_location=Id}]}`. Zotonic modules can listen to this message and return this information. This can be very handy if you have to create dynamic interfaces based on what modules are enabled:

```
{% for item in m.notifier.map[{footer_navigation current_location=id}] %}
    {% include "footer_item.tpl" item=item %}
{% endfor %}
```

m_oauth_app

- Module: *mod_oauth* (page 306)

Not yet documented.

m_oauth_perms

- Module: *mod_oauth* (page 306)

Not yet documented.

m_page

- Module: core

Todo

Not yet documented.

m_persistent

- Module: core

This implements persistent storage of values between visits from the same browser.

Persistent storage is implemented using a cookie and database storage. The cookie contains the key for the stored values. If a visitor makes a request then the session will load the persistent values into the session process. These can then be accesses using `{{ m.persistent.yourkey }}` in templates (replace yourkey with the key you used to store the information).

The persistent cookie (`z_pid`) is only set if persistent information is saved. This can be done using calls zo `z_context:set_persistent/3` or the action action-persistent_set.

Retrieve a value with `z_context:get_persistent/2` or the *m_persistent* (page 423).

Note that the first time a persistent value also a cookie needs to be set. This cookie will be added to the *Context* passed to the *set_peristent* call. If this was a regular request (HTTP or Ajax) then the cookie will be returned in the request reply. If the request was a Websocket request then the user-agent will make a call back to the server to retrieve (and set) the new persistent cookie.

m_predicate

- Module: core

Retrieve information about predicates. Predicates are the *labels* on edges (connections between resources) that give meaning to an edge. An example is the predicate "author" which refers to the authors of an article. Predicates form together with the referring and the referred page a triple `{subject, predicate, object}`.

Each predicate has a list of valid subject categories and valid object categories. This is used to filter the list of predicates in the admin edit page, and also to filter the list of found potential objects when making a connection.

A full predicate definition can be fetched by name or id with:

```
{{ m.predicate.author }}
{{ m.predicate[104] }}
```

Which both return a property list with information about the predicate. The property list contains all page properties and the properties: "pred" which is the atomic predicate name, "subject" which is a list of valid subject categories and "object" with is a list of valid object categories.

The following m_predicate model properties are available in templates:

Property	Description	Example value
all	Return a property list of all predicates. Keys are the atomic predicate name, values are property lists with information about the predicate. The property list contains all page properties and the properties: "pred" which is the atomic predicate name, "subject" which is a list of valid subject categories and "object" with is a list of valid object categories.	`[{about, [{pred,,about}, {subject,[104]}, {object,[]}, {id,300}, ... ]`
object_category	Used to derive the list of valid object categories for a predicate. Example usage: `m.predicate.object_category.author` Note: Each id is a 1-tuple.	`[{104}, ... ]`
subject_category	Used to derive the list of valid subject categories for a predicate. Example usage: `m.predicate.subject_category.author` Note: Each id is a 1-tuple.	`[{674}, ... ]`

m_req

- Module: core

This model gives access to the request variables from within a template.

Sometimes you need direct access to request variables in your template. The m_req model is meant for this. It exposes some values from the Webmachine request record.

Fetching a single value

You can fetch individual values by key, for example:

```
{{ m.req.host|escape }}
```

Viewing all request variables

Use the *print* (page 454) tag to get a complete overview of all request variables:

```
{% print m.req|make_list %}
```

This will show something like:

```
[{method,'GET'},
 {version,{1,1}},
 {peer,"127.0.0.1"},
 {is_ssl,false},
 {host,"test.dev:8000"},
 {raw_path,"/en/page/1234?foo=bar"},
 {path,"/en/page/1234"},
 {qs,[{"foo","bar"}]},
 {referrer,"http://test.dev:8000/"},
 {user_agent,"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_5) AppleWebKit/601.4.4␣
↪(KHTML, like Gecko) Version/9.0.3 Safari/601.4.4"},
 {is_crawler,false},
 {req_id,525158920},
 {headers,[{"accept",
            "text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8"},
           {"accept-encoding","gzip, deflate"},
           {"accept-language","en-us"},
           {"cache-control","max-age=0"},
           {"connection","keep-alive"},
           {"cookie",
            "z_logon=; z_sid=LopWHBmHXCs94virnboZhBHLKV6m1Cga; z_ua=c%3Ddesktop%26u
↪%3D1%26t%3D1%26w%3D1920%26h%3D1200"},
           {"dnt","1"},
           {"host","test.dev:8000"},
           {"referer","http://test.dev:8000/"},
           {"user-agent",
            "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_5) AppleWebKit/601.4.4␣
↪(KHTML, like Gecko) Version/9.0.3 Safari/601.4.4"}]},
 {ua_class,desktop},
 {ua_props,[{is_user_select,true},
            {has_pointer,true},
            {is_desktop,true},
            {is_crawler,false},
            {is_wireless_device,false},
            {is_tablet,false},
            {ajax_support_javascript,true},
            {device_os,<<"-">>},
            {displayWidth,1920},
            {displayHeight,1200},
            {inputDevices,<<"-">>},
            {parentId,<<"generic-">>},
            {model,<<"browser">>},
            {vendor,<<"desktop">>},
            {id,<<"desktopDevice">>}]},
 {timezone,<<"UTC">>},
 {language,en}]
```

Please note that all values are raw and not escaped, take care to escape the values before you use them in your templates, using the *escape* (page 377) filter.

The *make_list* (page 387) filter is used to force the evaluation of the model; otherwise it would just print `{m, req,undefined}`.

m_rsc

- Module: core

The main resource model, which is the central part of the *Zotonic data model* (page 3). This model provides an interface to all resource ("page") information. It also provides an easy way to fetch edges from pages without needing to use the *m_edge* (page 417) model.

Properties of the resource model

A resource has the following properties accessible from the templates:

Property	Description	Example value
id	Id of the page, an integer.	42
title	Title of the page. Returns a binary.	<<"Breaking News">>
short_title	Short title of the page. Used in menus. Returns a binary.	<<"News!">>
summary	Summary of the page. Returns a binary or undefined.	<<"Page summary.">>
body	The HTML body of a page. Returns a binary or undefined.	<<" Hello ">>
date_start	Start date when the page has a period. Examples are events or the birth date of a person. Returns a datetime tuple or undefined.	{{2008,12,10},{15,30,00}}
date_end	End date when the page has a period. Returns a datetime tuple or undefined. When there is a start date then there is also an end date.	{{2009,12,5},{23,59,59}}
name	Unique name of the page. Returns a binary or undefined. Valid characters are a-z, 0-9 and _	<<"page_home">>
page_path	Unique path of the page, used for url generation. Returns a binary or undefined. Valid characters are a-z, 0-9, / and -	<<"/">>
is_page_path_multiple	Allow the page to be served on multiple URLs	false

page_url	The url of the page. Derived using the page's category, the page id and its slug. Returns a non flattened list. Returns the binary page_path when it is set.
	The additional parameter `with` can be used to pass extra (optional) query arguments to the url, for instance:
	`{{ id.page_url with t=now|date:"U" }}`
	`{{ id.page_url with t="new" u=m.acl.user.id }}`
		<<"/blog/42">>

page_url_abs	The absolute url of the page. Same as `page_url` but then with added protocol, hostname and port.	<<"http://example.org/blog/42">>

Continued on next page

Table 5.2 – continued from previous page

Property	Description	Example value
default_page_url	The page without considering its page_path setting.	<<"/page/42/my-slug">>
is_authoritative	Whether this page originated on this site or is imported and maintained on another site. Return a boolean.	true
uri	The absolute unique uri of this resource. Refers to the "id" dispatch rule for authoritative (local) resources. Returns a binary.	<<"http://example.com/id/42">>
category_id	Id of the category the page belongs to. Returns an integer.	102
category	Category record the page belongs to. Returns a property list.	[{id,102},{parent_id,undefined], ... ,{name, <<"person">>}]
seo_noindex	Whether to let search engines index this page. Returns a boolean or undefined.	false
seo_title	Title for on top of the browser window. Returns a binary or undefined.	<<"Welcome Title">>
slug	Slug used for url generation, appended to page urls. Binary or undefined. Valid characters are a-z, 0-9 and -	<<"the-world-is-flat">>
seo_keywords	Keywords for search engine optimization. List of keywords separated with commas. Returns a binary or undefined.	<<"world, model, flat">>
seo_desc	Page description for search engines. Returns a binary or undefined.	<<"The truth about the world's shape">>
is_me	Check if this page is the current user's person page. Returns a boolean.	false
is_visible	Check if this page is visible for the current user. Returns a boolean.	true
is_editable	Check if this page is editable by the current user. Returns a boolean.	false
is_linkable	Check if this page can be connected to another page. Returns a boolean.	false
is_ingroup	Check if the current user is a member of the group the page belongs to. Returns a boolean.	true
exists	Check if the page exists. Useful when checking if a named page is present or not. Returns a boolean.	true

Continued on next page

Table 5.2 – continued from previous page

Property	Description	Example value
is_a	Returns a list of the category hierarchy the page belongs to. The list is suitable for indexing with category atoms. Example usage: `{{ m.rsc[id].is_a.article }}`	[{text,true}, {article,true}]
is_cat	Direct check if a page is a certain category. More efficient then is_a. Example usage: `{{ m.rsc[id].is_cat.person }}`	true
is_featured	If featured checked or not. Returns a boolean	false
is_protected	If this page is protected from deletion. Returns a boolean. Resources are protected by a simple table called `protect` that prevents accidental deletions of rsc records. It does this by having a foreign key constraint that prohibits the deletion of the referred rsc record.	false
is_dependent	If set to *true* then this page should only exist if there are incoming edges to this page. This flag is checked when an edge is deleted.	true
is_published	If this page has been published. Returns a boolean	true
publication_start	Start date of the publication period. Returns a datetime tuple.	{{2009,12,24},{9,0,0}}
publication_end	End date of the publication period. Returns a datetime tuple.	{{9999,8,17},{12,0,0}}
is_published_date	If this page is published and the current date/time is within the set publication_start/end range. Note that no ACL checks are performed, use *is_visible* to check if a resource is visible for the current user.	true
visible_for	Visibility level. Returns an integer. 0 = world visible, 1 = only for logged on users, 2 = only for group members, 3 = only for current user. The actual meaning depends on the active ACL module.	0
content_group_id	Content group this resource belongs to. Defaults to the id of the content group named *default_content_group* or *system_content_group* for resources within the *meta* category. See *mod_content_groups* (page 288)	31415

Continued on next page

Table 5.2 – continued from previous page

Property	Description	Example value
o	Used to access the objects of page: the pages this page refers to. Returns a function which should be indexed with the edge's predicate name (atom). When indexed the function will return a list of integers. Example usage: `{{ m.rsc[id].o.author[1].title }}` This returns the first author that is linked from this page.	fun(Predicate,Context)
s	Access the subjects of a page: the pages that are referring to this page. Returns a function which should be indexed with the edge's predicate name (atom). When indexed the function will return a list of integers. Example usage: `{{ m.rsc[id].s.author[1].title }}` This returns the first article that links to me with a author connection.	fun(Predicate,Context)
op	Returns a list of all predicates on edges from this page. The predicates are atoms.	[about, related]
sp	Returns a list of all predicates on edges to this page. The predicates are atoms.	[author]
predicates_edit	Returns a list of all allowed predicates from this page. Used for editing the page. Returns a list of predicate ids (in contrast with the atoms of op and sp).	[308,300,304,303,302,300]
media	Return a list of all media ids connected to the page. The media are connected with the predicate "depiction".	[842,3078]
medium	Return a property list describing the file or medium attached to the page. A medium record is present for pages that are an image, video etc. Returns undefined when there is no medium defined. See the model m_media for more information.	[{id,512}, {filename, <<"2009/1…">>, …]

Continued on next page

Table 5.2 – continued from previous page

Property	Description	Example value
depiction	Return the medium record that can be used for the image of a page. Either returns a medium page attached to the page or the medium record of the page itself. When no medium is found then undefined is returned.	[{id,512}, {filename, <<"2009/1...">>, ...]
email	E-mail address. Returns a binary or undefined.	<<"me@example.com">>
website	URL of a website. Returns a binary or undefined.	<<"http://zotonic.com">>
is_website_redirect	Tell *controller_page* to redirect to the website URL instead of showing the HTML page.	false
phone	Phone number. Returns a binary or undefined.	<<"+31201234567">>
phone_alt	Alternative phone number. Returns a binary or undefined.	undefined
phone_emergency	Phone number to call in emergencies.	<<"112">>
address_street_1	Address line 1. Returns a binary or undefined.	
address_street_2	Address line 2. Returns a binary or undefined.	
address_city	City part of address. Returns a binary or undefined.	
address_postcode	Postcode part of address. Returns a binary or undefined.	
address_state	State part of address. Returns a binary or undefined.	
address_country	Country part of address. Returns a binary or undefined.	
mail_street_1	Mailing address line 1. Returns a binary or undefined.	
mail_street_2	Mailing address line 2. Returns a binary or undefined.	
mail_city	City part of mailing address. Returns a binary or undefined.	
mail_postcode	Postcode part of mailing address. Returns a binary or undefined.	
mail_state	State part of mailing address. Returns a binary or undefined.	
mail_country	Country part of mailing address. Returns a binary or undefined.	
name_first	First name of person. Returns a binary or undefined.	
name_middle	Middle name of person. Returns a binary of undefined.	
name_surname_prefix	Prefix for the surname of a person. Returns a binary or undefined.	<<"van der">>, <<"von">>
name_surname	Surname or family name of person. Returns a binary or undefined.	

Escaping

All strings that are stored inside resources are automatically *HTML escaped*. This means that these texts do not require any processing when they are being displayed in the browser, which causes major performance gains.

There are some fields in the resource that are exceptions to these rule, namely, the `body` field and any fields whose name ends in `_html`. These fields are assumed to contain HTML text and are sanitized on save instead of escaped.

Dates

Dates are stored as a standard Erlang date time tuple, for example `{{2008,12,10},{15,30,00}}`. Dates are stored and retrieved in UTC (universal time). When displaying a date, (e.g. with the *date* (page 369) filter), the date is automatically converted into the time zone of the site or that of the user.

Printing all properties of a resource

In your templates, you can loop over the properties of a resource like this:

```
{% for k,v in m.rsc[id] %}
  {{ k }} - {{ v }} <br/>
{% endfor %}
```

And also using the *print* (page 454) tag:

```
{% print m.rsc[id] %}
```

See also:

The Zotonic data model (page 3), *m_edge* (page 417), *m_media* (page 421), *m_rsc_gone* (page 431).

m_rsc_gone

- Module: core

This model tracks deleted resources (see *m_rsc* (page 426)). Its primary goal is to be able to determine if a resource never existed, has been deleted or has been replaced by another resource.

Information kept

Only very basic information of the deleted resource is kept in the `rsc_gone` table. It is enough for referring to a new location, giving correct errors or to determine who deleted a resource.

It is not enough to undelete a resource. The module *mod_backup* (page 286) retains enough information about past versions to be able to undelete a resource. Currently there is no support for an undelete.

Properties

Whenever a *m_rsc* (page 426) record is deleted some information from that resource is copied to the `rsc_gone` table.

The following properties are saved:

Property	Description	Example value
id	Id of the resource, an integer.	42
new_id	If the resource is replaced by another resource then this is the id of that other resource.	341
new_uri	If the resource is moved to another place on the web then this is the uri of the new location.	<<"http://example.com/hello">>
name	The name (if any) of the deleted resource.	<<"page_hello">>
uri	The uri of the authoritative source for the resource.	<<"http://foo.bar/hello">>
page_path	The page path (if any) of the deleted resource.	<<"/hello">>
is_authoritative	Whether the resource originated on this site or was imported and maintained on another site. Return a boolean.	true
creator_id	The id of the creator of the deleted resource.	1
created	The date the deleted resource was created.	{{2008,12,10},{15,30,00}}
modifier_id	The id of the user that deleted the resource.	2718
modified	The date the resource was deleted.	{{2012,12,5},{23,59,59}}

See also:

The Zotonic data model (page 3), *m_rsc* (page 426)

m_search

- Module: core

The m_search model provides access to different kinds of search queries for searching through models.

Most searches in Zotonic are implemented in the *mod_search* (page 308) module, searching through the `rsc` table in different kinds of ways.

Though, any module can implement a search by observing the `search_query` notification.

The search module is used inside templates. For example, the following snippet fetches the latest 10 modified pages in the "text" category:

```
{% for id in m.search[{latest cat="text" pagelen=10}] %}
    {{ m.rsc[id].title }}
{% endfor %}
```

Another example, searching for a text and requesting the second page with 20 results at a time:

```
{% for id, rank in m.search.paged[{fulltext text=query_string page=2 pagelen=20}]
↪%}
    {{ m.rsc[id].title }}
{% endfor %}
```

See also:

Search (page 27), the scomp-pager scomp, and the *mod_search* (page 308) module.

Implementing a custom search

Like stated, any module can implement a search by observing the `search_query` notification:

```
 observe_search_query({search_query, Req, OffsetLimit}, Context) ->
     search(Req, OffsetLimit, Context).

search({foo_table, [{bar, Bar}]}, _OffsetLimit, _Context) ->
  #search_sql{
      select="f.id",
```

```
      from="foo f",
      where="f.bar = $1",
      order="f.baz desc",
      args=[Bar],
      tables=[{foo,"f"}]
  };

search(_Other, _OffsetLimit, _Context) ->
    undefined.
```

Do not forget to add the last function that matches and returns `undefined` for any other search request. If you forget this, the notification fold will crash when using any other search query.

This can then be used in your template like this:

```
{% for id in m.search[{foo bar=123}] %}
... looping over all ids in the 'foo' table for which bar = 123
{% endfor %}
```

m_session

- Module: core

Todo

Not yet documented.

Todo

move this section somewhere?

visitor / visitor_cookie

Every visitor to a Zotonic web site gets a unique id. This id is stored as a cookie on the user agent. When the visitor returns to the site then he/she will be recognized by the cookie with the visitor id. This enables the possibility to remember the visitor's settings and preferences.

When a visitor logs on then the Zotonic can attach the user's visitor record to the user's rsc record. Zotonic can then also merge multiple visitor records into one. Enabling the possibility to remember preference changes between different user agents.

m_signal

- Module: *mod_signal* (page 311)

`m.signal`

Makes it possible to access the emitted signal in a template.

`m.signal[signal].type` - The type of the emitted signal

m_site

- Module: core

Retrieve information which is stored in the site's *config* files.

The site configuration is stored for each site in `user/sites/<sitename>/config` and files in `user/sites/<sitename>/config.d/`. Their syntax is equal to an Erlang property list, with unique keys.

Fetch a site configuration key

Example, fetching the site configuration key "hostname":

```
{{ m.site.hostname }}
```

Fetching all configurations

It is easy to loop over all site configurations:

```
{% for key, value in m.site %}
    {{ key }} -- {{ value }} <br />
{% endfor %}
```

Overriding config values

Zotonic has two places where a site's configuration is kept. One is in the site's config files, the other in the config table. The config table overrules any module settings from the config file, for rows where the *module* key of the config value is set to *site*.

Within the site configuration, you can override module-specific configuration: Module configurations are defined with a property key equal to the module name, with a property list as value. For example:

```
{mod_foo, [ {hostname, "localhost"} ]}
```

will put the default "hostname" setting of the imaginary `mod_foo` module to *localhost*.

m_survey

- Module: *mod_survey* (page 316)

Not yet documented.

m_tkvstore

- Module: *mod_tkvstore* (page 317)

Simple read-only interface to the typed key-value store of *mod_tkvstore* (page 317). To get a value from the store: use *m.tkvstore.type.key*, like this:

```
{% print m.tkvstore.keytype.examplekey %}
```

When the key is not a literal value but a variable or a number, use the following notation:

```
{% print m.tkvstore.keytype[keyvar] %}
```

To store data in a key, use `m_tkvstore:put/4`, as follows:

```
m_tkvstore:put(KeyType, KeyVar, Value, Context).
```

For instance, from within Erlang, let's store *edge data* for a given edge id, 3434:

```
Edgeid = 3434,
Value = <<"Hello">>,
m_tkvstore:put(edge_data, EdgeId, Value, Context).
```

Now, to retrieve it in the template, do the following:

```
{{ m.tkvstore.edge_data[3434] }}
```

This will output the string `Hello`.

Note that the value can be any type: not only a simple string but also a list, tuple, or any other Erlang composite type.

See also:

mod_tkvstore (page 317)

m_translation

- Module: *mod_translation* (page 317)

The m_translation model gives easy access to language and translation related information.

The following `m.translation` model properties are available in templates:

Property	Description
language	The current language.
language_list	The list of all configured languages.
language_list_enabled	The list of all enabled languages.

This is an example of the languages returned by `m.translation.language_list`:

```
[{en, [{is_enabled,true}, {language,<<"English">>}]},
 {fr, [{is_enabled,false}, {language,<<"Français">>}]},
 {nl, [{is_enabled,true}, {language,<<"Nederlands">>}]},
 {tr, [{is_enabled,true}, {language,<<"Türkçe">>}]}].
```

For example to list all enabled languages in a select box:

```
<select>
{% for code,props in m.translation.language_list_enabled %}
  <option value="{{ code }}" {% if m.translation.language == code %}selected{%␣
↪endif %}>{{ props.language }}</option>
{% endfor %}
</select>
```

Services

Services are the entry points to the Zotonic REST-based API. For API access, OAuth or session-cookies are used as authentication mechanism. JSON is used as output mechanism.

See also:

API services (page 51)

Returns a single array with every id of every *Resource* in the database:

```
$ curl http://localhost:8000/api/base/everything

[1,101,102,103,104,106,107,108,109,110,111,112,113,114,115,116,117,119,120,121,122,
↪123,300,301,303,...
```

The results are paged per maximum 100 items, supply a *page* argument to fetch another page than page number 1:

```
$ curl http://localhost:8000/api/base/everything?page=10

[4169,4170,4171,4172,4173,4174,4175,4176,4177,4178,4179,4180,4181,4182,4183,4184,
↪4185,4186,4187,4188,4189,4190,4191,4192,...
```

Due to access control restrictions it is possible that less than 100 ids are returned for a page.

Exports a "dump" of the given *resource*. A dump is a JSON object with *rsc*, *category*, *media*, *edges* sub-objects. When requesting the service:

```
http://localhost:8000/api/base/export?id=325
```

The following might be returned:

```
{
  "id": 325,
  "uri": "http://localhost:8000/en/id/325",
  "rsc": {
      "category_id": 106,
      "created": "2012-10-25 16:18:08",
      "creator_id": 1,
      "id": 325,
      "is_authoritative": true,
      "is_featured": false,
      "is_protected": true,
      "is_published": true,
      "modified": "2012-10-25 16:18:08",
      "modifier_id": 1,
      "name": "wordpress_1",
      "page_path": null,
      "pivot_geocode": null,
      "publication_end": "undefined",
      "publication_start": "2008-04-15 09:28:00",
      "slug": "a-somewhat-brief-history",
      "uri": null,
      "version": 6,
      "visible_for": 0,
      "date_end": null,
      "date_start": null,
      "title": "A (somewhat) brief history",
      "body": "<p>...</p>",
      "summary": "",
      "custom_slug": true,
      "installed_by": "mod_import_wordpress"
  },
  "edges": [
      {
          "predicate_id": 301,
          "predicate_name": "author",
          "object_id": 321,
          "seq": 1000000,
          "predicate_uri": "http://localhost:8000/en/id/301",
          "predicate_title": {
          "trans": {
              "en": "Author",
              "nl": "Auteur"
           }
          },
          "object_uri": "http://localhost:8000/en/id/321",
          "object_title": "jlueck"
      },
```

```
        ...
    ]
}
```

Returns a JSON object with information about the current user and the system:

```
http://localhost:8000/api/base/info
```

Returns a JSON object like this:

```
{
  "user": {
      "user_name": "Site Administrator",
      "user_id": 1
  },
  "site": {
      "zotonic_version": "0.9-dev",
      "language": "nl"
  }
}
```

Service URL:

```
/api/base/media_upload
```

Request method(s): POST

Upload media items into Zotonic. Pass in the *file* argument for the actual file. Because it's a file upload, the post payload should be *multipart/form-data* encoded (which is the standard for file uploads). Proper authorization is needed to use this API call, either through session cookie or using OAuth. The value returned is a single integer with the ID of the newly created media rsc.

Other arguments to this API call that can be passed in are: title, summary, body, chapeau, subtitle, website, page_path.

When upload succeeds it returns a JSON object like the following:

```
{"rsc_id": 123}
```

Where *rsc_id* is the id of the newly created *resource*.

In case of failure, a JSON message like the following is returned:

```
{"error": {"code": 403, "message": "Access denied"}}
```

Given an `id` as input argument, returns a JSON export of the resource's menu structure.

The JSON returned is a nested list of objects which each contain the pages `id`, `url` and, optionally, any `children`:

```
[
  {
      "id": 51253,
      "title": "About",
      "url": "/about"
      "children": [
          {
              "id": 51246,
              "title": "Our team",
              "url": "/about/team"
          },
          {
              "id": 51246,
              "title": "Our work ethics",
```

```
                "url": "/about/ethics"
            }
        ]
    },
    {
        "id": 51254,
        "title": "Contact",
        "url": "/contact"
    },
    {
        "id": 50497,
        "title": "Blog",
        "url": "/page/333/blog"
    }
]
```

Returns meta-information about all available API calls:

```
http://localhost:8000/api/base/meta
```

Returns a JSON object like the following:

```
[
    {
        "method": "search/search",
        "module": "mod_search",
        "service": "service_search_search",
        "title": "Search Zotonic resources.",
        "needauth": false,
        "http": "GET,HEAD"
    },
    {
        "method": "base/meta",
        "module": "mod_base",
        "service": "service_base_meta",
        "title": "Meta-information about all API calls.",
        "needauth": false,
        "http": "GET,HEAD"
    },
    ...
]
```

Retrieve a value from the Zotonic visitor record.

The visitor record's value for the `key` argument is returned:

```
http://localhost:8000/api/base/persistent_get?key=foo
```

might return:

```
{
    "key": "foo",
    "value": null
}
```

If the value is undefined; or

```
{
    "key": "foo",
    "value": "bar"
}
```

If the value was set.

See also:

service-persistent_set

Set a value in the Zotonic visitor record.

Required arguments are `key` and `value`.

http://localhost:8000/api/base/persistent_set?key=foo&value=bar

Returns a key/value tuple to confirm that the value has been set:

```
{
    "key": "foo",
    "value": "bar"
}
```

See also:

service-persistent_get

Remotely recompile and flush.

This GET requests performs a `z:m()` call which recompiles all of Zotonic's Erlang modules and all Erlang modules in all Zotonic sites.

When done recompiling, it calls `z:flush()` to flush the memo caches.

This call is mostly used for development purposes; for production use, *mod_development* (page 289) should not be enabled.

Search Zotonic's *resources* using the *Search* (page 27).

For instance, the API call:

```
http://localhost:8000/api/search?cat=text&text=test
```

Returns a JSON list of all resource ids of the *category* (page 5) `text` that contain the string `test`:

```
[320]
```

Adding `&format=simple` to the API call gives us a list of JSON objects:

```
[
  {
      "category": [
          "text"
      ],
      "id": 338,
      "preview_url": "http://example.dev:8000/image/2014/10/8/image_2014_09_09_19_
↪19_41.png%28800x800%29%28upscale%29%28CF8AD1D93AC1B8457F9AD6B9BA64C74F%29.jpg",
      "summary": {
          "trans": {
              "en": "English summary",
              "es": "Spanish summary"
          }
      },
      "title": {
          "trans": {
              "en": "English",
              "es": "Hola hola Espanol"
          }
      }
  }
]
```

Parameters

You can enter all parameters that are in the standard *Search* (page 27).

Besides, the following parameters exist:

limit

The number of results to return. Defaults to 20; maximum number to return per call is 1000. To return more, do multiple requests with the `offset` parameter.

offset

Start offset for the result set.

format

Either `ids` to return a plain array of rsc ids; or `simple` to return a list of JSON objects with the resource's *title*, *summary*, *category*, and *preview_url* (the first image).

See also:

Search (page 27)

Post a survey as JSON.

The JSON payload in the body of the request should be a JSON object with all values from the survey fields. The "id" argument in the URL specifies which form (survey resource) to submit.

Example:

```
curl -vv -H 'Content-type: application/json' -d '{"truefalse1": "no"}' http://
↪localhost:8000/api/survey/submit?id=100
```

When all is OK, the following result is returned:

```
{"result":"submitted"}
```

In case of an error, an error JSON is returned:

```
{"error":{"code":"syntax","message":"Syntax error: Missing fields: email"}}
```

When an extra query parameter is posted, `allow_missing=true`, the API call will never complain about missing fields but just submit the survey anyway.

Todo

Not yet documented.

Builtin Tags

When you are trying to find a tag used in a template, and it is not listed here, then check the list of all *Custom Tags* (page 457) as the scomps syntax is identical to the *Tags* (page 22) syntax.

_ (extended translation)

Select a translation from the arguments.

The extended translation tag `{% _ %}` is used for specifying translations in the template itself. This compared to the usual *translate* (page 456) tag `{_ ... _}` or `{{ _"text" }}` values which depends on separate gettext translations.

The arguments of the `{% _ %}` tag are the english text and the translations in different languages.

For example:

```
{% _ "Example"  nl="Voorbeeld"  fr="Exemple" %}
```

See also:

translate (page 456).

all catinclude

Include a template for all a resource's categories from all modules.

This is an extension on the *catinclude* (page 444) tag. It will include all templates with the given name, instead of the first one found. Templates are defined in modules, because of that multiple modules can define a template with the same name.

This catinclude extension will include all available templates in the same order as defined in *catinclude* (page 444). Where the templates per category will be rendered in the order of their module's defined priority. This is the order in which they are listed in the module admin page.

Examples of this mechanism can be found in *mod_admin* (page 275), for example the main menu and the category specific editing fields on the edit page.

An example usage:

```
{% all catinclude "hello.tpl" id arg="val" %}
```

Includes all templates with the base name *hello.tpl* for the id's category hierarchy.

For example, in the case of a *news* article:

- all templates with the name .hello.news.tpl.
- all templates with the name .hello.article.tpl.
- all templates with the name .hello.text.tpl.
- all templates with the name .hello.tpl.

See also:

tags *catinclude* (page 444) and *all include* (page 441).

all include

Call all modules to include a certain template.

This is an extension on the *include* (page 451) tag. It will include all templates with the given name, instead of the first one found. Templates are defined in modules, because of that multiple modules can define a template with the same name.

For example when you have two modules (*mod_a* and *mod_b*), both with the template *_name.tpl*. When the template in *mod_a* is defined as:

```
this is mod_a's {{ hello }}
```

and in mod_b as:

```
this is mod_b's {{ hello }}
```

then the tag:

```
{% all include "_name.tpl" hello="world" %}
```

Will output:

```
this is mod_a's world
this is mod_b's world
```

The modules will be called in the order of their defined priority. This is the order in which they are listed in the module admin page.

Examples of this mechanism can be found in *mod_admin* (page 275), for example the main menu and the category specific editing fields on the edit page.

Another example is the *_html_head.tpl* template which is included from the template-base template and allows all modules to add HTML to the head of a generated HTML page.

See also:

tag *include* (page 451).

autoescape

Automatically apply HTML escaping to values.

The *autoescape* tag controls the current auto-escaping behavior. This tag takes either *on* or *off* as an argument and that determines whether auto-escaping is in effect inside the block.

When auto-escaping is in effect, all variable content has HTML escaping applied to it before placing the result into the output (but after any filters have been applied). This is equivalent to manually applying the escape filter to each variable.

Example:

```
{{ value }}
{% autoescape on %}
  {{ value }}
{% endautoescape %}
```

When the variable value contains *Hello* then this will output:

```
<b>Hello</b>
&lt;b&gt;Hello&lt;/b&gt;
```

block

Define a block in a template and overrules a block from an inherited template.

The *block* tag is used for replacing blocks in inherited templates.

For example, when we have a template *base.tpl*, in which we define a block called *name*:

```
Hello {% block name %}my{% endblock %} world.
```

And we define a second template, *page.tpl*, which *extends* (page 445) the first template:

```
{% extends "base.tpl" %}
{% block name %}Peter's{% endblock %}
```

Then the result of rendering *page.tpl* will be:

```
Hello Peter's world.
```

If we do not include the block definition, so *page.tpl* just contains the *extends* (page 445) tag:

```
{% extends "base.tpl" %}
```

then the output will be:

```
Hello my world.
```

The name of a block must be a valid identifier, consisting of alphanumeric characters (a-z, 0-9) and the underscore charater.

See also:

extends (page 445) and *overrules* (page 454).

cache

Cache a frequently used block for later reuse.

Cache the output of the enclosed block. The block can be named. Cache blocks with the same name will use each others cached entries, when the cache entry is still valid.

Example:

```
{% cache 3600 now %}Local time is {% now "Y-m-d H:i:s" %}{% endcache %}
```

This caches the output of the block for an hour. The name for the cache is "now".

The cache duration and name are optional. The default cache duration is 0 (zero) seconds, which gives parallel rendering protection (see below) though will not store the result in the cache.

The cache tag protects against the situation where parallel requests need to render the same block at the same time. Instead of all the processes rendering the block in parallel, one process will render the block and share the result with the other—waiting—processes.

Example, prevent parallel rendering (slam dunk) by non logged on visitors:

```
{% cache if_anonymous %} insert complicated page here {% endcache %}
```

Besides the duration and the cache name the `{% cache %}` tag also accepts the following arguments:

Argument	Description	Example
vary	This argument can be used multiple times. Cache blocks with different vary arguments have different cache keys. The arguments are assumed to be keys in the cache. When one of the vary keys is updated or invalidated then the cached block will be invalidated.	vary=myid
cat	Category the cached block depends on. This argument can be used multiple times for specifying multiple categories. The categories are not added to the cache key, only added as cache dependencies.	cat="news"
if	Only cache the block when the argument evaluates to true.	if=can_cache
if_anonymous	Only cache the block when the current visitor is not logged on (i.e. an anonymous visitor)	if_anonymous
visible_for	Sets the access control user for rendering the block. With this you can force to only show public items for logged on users. Valid values are "user", 3, "group", 2, "community", 1, "world", "public", 0	visible_for="world"

call

Call an Erlang function.

The `{% call %}` tag is used to call the `render/3` function of the module specified by the argument.

For example:

```
{% call mymodule a=1 b=2 %}
```

Will call `mymodule:render([{a,1}],[{b,2}], TemplateVariables, Context)`. Where *TemplateVariables* is the property list with all template variables and *Context* is the current Zotonic request *context*.

The *render/2* function must return either `{ok, IoList}` or `{error, Reason}`.

If `{error, Reason}` is returned then the `Reason` is rendered as `io_lib:format("~p", [Reason])`

For compatibility with Django it is possible to pass a single value instead an argument list:

```
{% call mymodule with value %}
```

This will call `mymodule:render(Value, TemplateVariables, Context)`.

catinclude

Include another template based on the category of a resource. The include tag is replaced with the contents of the included template file. You can give arguments to the included template, they will be assigned as variables in the context of the included template.

Example:

```
{% catinclude "hello.tpl" id %}
```

Assuming that the resource whose id is the value of the template variable *id* is a news article then catinclude will consider the following templates:

```
hello.news.tpl
hello.article.tpl
hello.text.tpl
hello.tpl
```

This because *news* is a subcategory of *article*, which is a subcategory of *text*. When one of the previous templates is not found then the base template *hello.tpl* is tried. The catinclude tag will only include the first file it finds, and stops after having found a file to include.

When the resource has a unique name (the *name* property is set), this property is also considered for the catinclude lookup, before the category-based template names. So when the resource has *name* set to *foobar*, it will first look for `hello.foobar.tpl`, then for `hello.news.tpl`, etc.

Unlike Django the template name must be a string literal, variables are not allowed.

The tag accepts extra arguments, which will be passed as template variables to the included template. The inclusion is always done at runtime, because the selected template depends on the category of the referenced resource.

The resource id will be available in the included template as the variable *id*.

Instead of passing an id, you can also pass in a list of category names which are to be search for. These names need to be atoms, like this:

```
{% catinclude "hello.tpl" [`text`, `article`] %}
```

This will search for the following templates, in order:

```
hello.article.tpl
hello.text.tpl
hello.tpl
```

See the *include* (page 451) for caching options and argument handling.

See also:

all catinclude (page 441), which is useful to include multiple templates.

comment

Ignore part of a template.

Everything inside a `{% comment %}` block is not output.

Example:

```
This will show.
{% comment %} And this will not show {% endcomment %}
```

This will output:

```
This will show.
```

An alternative to the `{% comment %}` tag is to use the `{# ... #}` construct:

This will show. {# And this will not show #}

The big advantage of this notation is that the contents of the `{# ... #}` construct don't need to be grammatically correct, as they will not be parsed. The contents of a `{% comment %}` block must be correct as they will be parsed by the template compiler.

cycle

Rotate through a list of values.

Rotates through a list of values and outputs them. `{% cycle %}` Is used within a `{% for %}` loop.

Example:

```
{% for a in [1,2,3,4] %}{% cycle "bleu" "blanc" "rouge" %} {% endfor %}
```

Will output "bleu blanc rouge bleu ".

The list values can be either a string literal, a number literal, a variable or an automatic id (`#name`).

Note: You can not apply filters to the cycle values.

---

extends

Inherit markup from another template.

Signal that this template extends another template. The extends tag must be the first tag in a template that inherits from another template.

Note: A template that extends another template contains only the extends tag and *block* (page 442) tags.

Example:

```
{% extends "base.tpl" %}
```

All named blocks in this template will replace the similar named blocks in the template *base.tpl*.

Unlike Django the template name must be a string literal, variables are not allowed.

See also:

block (page 442), *inherit* (page 452) and *overrules* (page 454).

filter

Filter the contents of a block through variable filters.

Filters can also be piped through to each other.

Example:

```
{% filter escape | lower %}
The text will be lowered and escaped. So you can use <, > and & without any␣
↪problems.
{% endfilter %}
```

New in version 0.8.

firstof

Not implemented, but exists in Zotonic for forward compatibility with future ErlyDTL and Django versions.

for

Loop over multiple values in a list or search result.

`{% for %}` loops over a list of values. For example to loop over a list of colors:

```
{% for color in ["bleu", "blanc", "rouge"] %}{{ color }}{% endfor %}
```

This will output "bleublancrouge".

Or to show the latest ten news article titles:

```
{% for id in m.search[{latest cat="news" pagelen=10}] %}
  {{ m.rsc[id].title }}
{% endfor %}
```

The argument list can also contain tuples or sublists, in that case you can assign all parts of the list items to different variables at once:

```
{% for nr,color in [ [1,"bleu"], [2,"blanc"], [3,"rouge"] ] %}
  {{ nr }}: {{ color }}
{% endfor %}
```

This will output:

```
1: bleu
2: blanc
3: rouge
```

The for loop sets a number of variables available in the loop:

Variable	Description
forloop.counter	The current iteration of the loop (1-indexed)
forloop.counter0	The current iteration of the loop (0-indexed)
forloop.revcounter	The number of iterations from the end of the loop (1-indexed)
forloop.revcounter0	The number of iterations from the end of the loop (0-indexed)
forloop.first	True if this is the first time through the loop
forloop.last	True if this is the last time through the loop
forloop.parentloop	For nested loops, this is the loop "above" the current one

For ... empty

`{% for %}` can take an optional `{% empty %}` clause that will be displayed if the given list is empty.

For example:

```
{% for id in m.search[{latest cat="news" pagelen=10}] %}
  {{ m.rsc[id].title }}
{% empty %}
  Sorry, there is no news.
{% endfor %}
```

This will output "Sorry, there is no news." when the "latest" search did not find any pages within the category "news".

if

Show a block if the condition is true.

The `{% if %}` tag evaluates a variable and if the result is true (boolean true, number unequal to zero, non empty string or a non empty list) then the contents of the if-block are output. When the if-variable test fails then the optional `{% elif %}` blocks are evaluated. When the if and all optional elif variable tests fail, the optional `{% else %}` block content is output.

Example:

```
{% if genre == "pop" %}
  Popular music.
{% elif genre == "classical" %}
  Classical music.
{% elif genre == "jazz" %}
  Jazz
{% else %}
  The genre isn't pop, classical or jazz.
{% endif %}
```

An `{% if %}` and `{% elif %}` tag can have an "and" or "or" expression as argument:

```
{% if person_list and show_persons and full_moon %}
  There are persons that we can show during full moon.
{% endif %}
```

Or for example:

```
{% if new_moon or daytime %} Guess you can't see the moon. {% endif %}
```

It is also possible to mix "and" and "or" in one expression, so this is a valid:

```
{% if full_moon or daytime and cloudy %}
```

The "not" operator can be used to negate a boolean value:

```
{% if full_moon or daytime or not clearsky %}
```

Note: Besides the `{% elif %}` tag we also support the alias `{% elseif %}`.

if-with

The `if` is often combined with the `with` tag. For example:

```
{% with m.search[{latest cat=`news` pagelen=10}] as result %}
    {% if result %}
        <h3>{_ Latest news _}</h3>
        <ul>
          {% for id in result %}
            <li><a href="{{ id.page_url }}">{{ id.title }}</a></li>
          {% endfor %}
        </ul>
    {% endif %}
{% endwith %}
```

To make this easier it is possible to combine the `if` and `with` tags in a single expression:

```
{% if m.search[{latest cat=`news` pagelen=10}] as result %}
    <h3>{_ Latest news _}</h3>
    <ul>
      {% for id in result %}
        <li><a href="{{ id.page_url }}">{{ id.title }}</a></li>
      {% endfor %}
    </ul>
{% endif %}
```

The `as` can also be used in the `elif` expressions:

```
{% if expression1 as x %}
  ...
{% elif expression2 as y %}
  ...
{% else %}
  ...
{% endif %}
```

See also:

ifequal (page 448) and *ifnotequal* (page 449).

ifchanged

Not implemented, but exists in Zotonic for forward compatibility with future ErlyDTL and Django versions.

ifequal

Show a block when two values are equal.

The `{% ifequal %}` tag tests if its two arguments are equal. If so then the contents of the `{% ifequal %}` block are output, otherwise the contents of the optional `{% else %}` block are output.

For example:

```
{% ifequal value 5 %}
  Value is equal to 5.
{% else %}
  Value is {{ value }} which is not 5.
{% endifequal %}
```

It is only possible to compare arguments that are variables (with optional filters) or constants. Examples of constants are numbers, strings or lists.

See also:

if (page 447) and *ifnotequal* (page 449).

ifnotequal

Show a block when two values are not equal.

The `{% ifnotequal %}` tag tests if its two arguments are unequal. If so then the contents of the `{% ifnotequal %}` block are output, otherwise the contents of the optional `{% else %}` block are output.

For example:

```
{% ifnotequal value 5 %}
  Value is {{ value }} which is not 5.
{% else %}
  Value is 5.
{% endifnotequal %}
```

It is only possible to compare arguments that are variables (with optional filters) or constants. Examples of constants are numbers, strings or lists.

See also:

if (page 447) and *ifequal* (page 448).

image

Show a still image using a `<img />` element.

The `{% image %}` tag is used to generate an HTML `<img />` element for a media resource. The image will be automatically resized to the desired size and filters. See also the `{% media %}` tag for handling video.

For example:

```
{% image "cow.jpg" width=200 height=200 crop %}
```

This will generate an image tag for the image "cow.jpg" (in the files/archive directory) of 200x200 pixels. The image will be resized and cropped to the requested size. The image tag will be something like (the checksum will vary per sign key):

```
<img src="/image/cow.jpg%28200x200%29%28crop%29%28981E1AD8DA60381D00C094F0EE1311F5
↪%29.jpg" width="200" height="200" />
```

The file argument can be one of the following:

- filename relative to the archive folder ("cow.jpg" is always present)
- resource id of a resource with attached file (mostly of the category "media")
- property list of a resource's medium record

The following arguments/filters can be specified:

Argument	Description	Example
width	The maximum width of the image.	width=200
height	The maximum height of the image.	height=200
mediaclass	The media class of the image. See *Media classes* (page 24).	mediaclass=”thumb”
background	The background color for transparent image parts. See ImageMagick colors for how to specify the RGB color.	background=”white”
removebg	Removes the image background. Accepts an optional fuzziness parameter (range 0..100).	removebg removebg=50
blur	Blur the image, making it less sharp. See ImageMagick blur for valid argument values.	blur=”20x8”
crop	Crop the image, the resulting image will be exactly the size specified in the *width* and *height* arguments. Which part of the image will be cropped depends on the value of the crop argument. When no argument is given to crop, it defaults to the center point of the image, unless there is a cropping center point defined in the media item. Other options are: *north*, *north_east*, *east*, *south_east*, *south*, *south_west*, *west*, *north_west* and *center*. When the “crop” argument is a string of the form “+x+y”, this coordinate is taken as the ‘center of gravity’ of the crop, to assure that the given point is in view after the image is cropped. This point is given in image coordinates (unscaled), and is relative to the top left of the image, so `crop="+0+0"` is the same as saying `crop="north_east"`.	crop crop=”south” crop=”+100+100”
extent	Make the image fit the requested dimensions by adding whitespace using this procedure: Resize the image so that it fits inside the width/height box; then extent the image with a white background so that it is centered and exactly the size of the box.	extent
upscale	Forces the image to scale up to the requested dimensions.	upscale
flip	Flip the image. Left and right will be mirrored.	flip
flop	Flop the image. Top and bottom will be mirrored.	flop
grey	Make the image greyscale.	grey
lossless	Controls whether resized images should become JPG (lossless=false) or PNG images (lossess=true). When set to *auto*, PNG images will stay PNG images when resized, as PNG images usually contain graphics, which tend to look bad when encoded as JPG. Defaults to *false*.	lossless=‘true‘ lossless=‘auto‘ lossless=‘false‘

See also *Media classes* (page 24) for some options that are only available in *mediaclass* files.

See also:

image_url (page 451), *Media classes* (page 24) and *media* (page 453).

image_url

Generate the url of a still image.

The `{% image_url %}` tag is used generate the url of an image. `{% image_url %}` accepts all parameters of the `{% image %}` tag but only outputs the url and not the `<img />` element to display it.

See also:

image (page 449).

include

Include another template. The include tag is replaced with the contents of the included template file. You can give arguments to the included template, they will be assigned as variables in the context of the included template.

Example:

```
{% include "_hello.tpl" name="Peter" %} world.
```

When *_hello.tpl* contains the text:

```
Hello {{ name }}'s
```

Then this will output the text `Hello Peter's world.`.

When the template name is a string literal then the template will be inlined. When it is an expression then the template will be included during runtime.

New in version 0.9.1: Added the *optional* keyword.

If the included template is not required, a *optional* keyword may be used:

```
{% optional include "might-not-exist.tpl" %}
```

Note:

About unique ids Automatically generated ids (`{{ #name }}`) are unique within an included template and do not clash with similarly named ids in the including template.

With keyword For compatibility with DTL we accept the optional *with* keyword betwen the template name and the arguments:

```
{% include "_hello.tpl" with name="Peter" %}
```

See also:

all include (page 441) and *catinclude* (page 444).

Caching of the included template

The output of the included template can be cached. This is useful when rendering the template takes considerable time, for example when the template shows a list of recent news items, which comprises a query, fetching and rendering a list of news items. To cache such a list:

```
{% include "recent_news_items.tpl" maxage=3600 %}
```

Caching is enabled by defining one of the caching arguments:

Argument	Description	Example
maxage	The maximum time the output can be cached, in seconds. Specifying `0` for the maximum age does not cache the output but does protect agains slam dunks, multiple requests rendering the same template at the same time will share the output of the rendering.	`maxage=3600`
vary	Dependency keys for the cached output. If a cache key with the same name is flushed or invalidated then the cached output of this template is also invalidated. You can use category names here.	`vary="news"`
sudo	If supplied then access control is disabled whilst rendering the included template. This will show any content not visible for the current user. Use with care.	`sudo`
runtime	If supplied then the included template is not inlined but included during evaluation of the calling template. Only the supplied arguments are available as variables in the included template.	`runtime`

New in version 0.6: Added the *sudo* option.

inherit

Include the markup of an extended template into the extending template.

Say you have a template `hello.tpl` containing:

```
{% block test %}
This is content from hello.tpl
{% endblock %}
```

And in your site you have a `world.tpl` template, defined as:

```
{% extends "hello.tpl" %}
{% block test %}
First line
{% inherit %}
This is more content from world.tpl
{% endblock %}
```

Then, the result of rendering the template `world.tpl` will be:

```
First line
This is content from hello.tpl
This is more content from world.tpl
```

See also:

block (page 442), *extends* (page 445) and *overrules* (page 454).

javascript

Adds javascript that will be run after jQuery has been initialized. In dynamic content it will run after the DOM has been updated with the template where the javascript was defined.

Example:

```
{% javascript %}
  ...
{% endjavascript %}
```

lib

Combine css and javascript includes in a single request.

Generates a `<link />` or `<script />` element for including the given libraries. This combines all css and javascript files in one request, saving multiple roundtrips to the server.

Example:

```
{% lib
      "css/zp-compressed.css"
      "css/zp-admin.css"
      "css/zp-wysiwyg.css"
      "css/zp-dialog.css"
      "css/zp-formreplace.css"
      "css/zp-finder.css"
      "css/zp-growl.css"
  %}
```

Will output:

```
<link href="/lib/css/zp-compressed~zp-admin~zp-wysiwyg~zp-dialog~zp-formreplace~zp-
↪finder~zp-growl~63417066183.css" type="text/css" media="all" rel="stylesheet" />
```

The number at the end is the Unix modification time of the most recently changed file.

The *lib* tag supports optional arguments to control the resulting html tag:

Option	Default	Description
use_absolute_url	false	If true, prefix the generated URL with "http://{hostname}/".
title	<empty>	Specify a value for the title attribute of the link tag.
media	"all"	Specify value for the media attribute of the link tag.
rel	"stylesheet"	Specify value for the rel attribute of the link tag.

See also:

mod_development (page 289).

load

Loads the given custom tags for use in the templates. Normally not needed, as custom tags are automatically loaded through the module system.

See also:

Custom Tags (page 457)

media

Show embed, video or audio media.

The `{% media %}` tag is similar to the *image* (page 449) tag. It accepts the same arguments but where `{% image %}` is guaranteed to give an still image, `{% media %}` can also generate embed, video or audio elements.

The `{% media %}` tag is not implemented in the core of Zotonic. It depends on modules implementing the tag which arguments and media formats are accepted.

An example of a module using `{% media %}` is *mod_video_embed* (page 320) which enables the use of embed code from sites as youtube and vimeo. Mod_video_embed will echo the embed code for a `{% media %}` tag and output a still image for an `{% image %}` tag.

See also:

image (page 449).

now

Show the current date and time.

Displays the current local date and time, formatted according to the given date format string.

Example:

```
{% now "Y-m-d" %}
```

Did output "2008-12-10" on december 10, 2008.

There is also a variable called `now`, which holds the current date:

```
{{ now|date:"Y-m-d" }}
```

Is equivalent to using the `{% now %}` tag.

See also:

the *date* (page 369) filter for the possible format characters.

overrules

Inherit markup from like named template in another module.

Signal that this template extends a template with the same name in a module with lower priority.

The *overrules* tag must be the first tag in the template.

Note: A template that overrules another template contains only the *overrules* (page 454) tag and *block* (page 442) tags.

Example, say a template "page.tpl" contains the following:

```
{% overrules %}
{% block title %} My new title {% endblock %}
```

All named blocks in this template will replace the similar named blocks in the template *page.tpl* that is "next in line" to be used.

This is useful if you want to use a template from a module, and the template is mentioned in (for example) a dispatch rule. Now you can overrule and extend that template in your own modules without changing the dispatch rules or the original module.

Make sure your module has a higher priority (lower number) than the module containing the overruled template.

See also:

block (page 442), *inherit* (page 452) and *extends* (page 445).

print

Show the contents of a value expression.

The `{% print %}` tag is used to dump the contents of a variable in a HTML safe way. It is very useful for debugging and inspecting variables during template development.

For example:

```
{% print value %}
```

When value is "Hello world!" then the example above returns the output:

```
<pre>
&lt;b&gt;Hello&lt;/b&gt; world!
</pre>
```

It can also print complex values like nested lists and tuples, for which it uses Erlang's `io_lib:format/2` function.

See also:

scomp-debug

raw

Make a literal section which does not interpret tags.

The `{% raw %}` tag takes everything between the `{% raw %}` and `{% endraw %}` without interpretation. It is useful for surrounding javascript or pieces of code with (for example) `{%` in it.

Example:

```
This echos: {{ a }}
{% raw %}
This does not echo {{ a }}
{% endraw %}
{{ a }}
```

Will output:

```
This echos: hello
This does not echo {{ a }}
hello
```

regroup

Not implemented tag, for forward compatibility with future ErlyDTL and Django versions.

spaceless

Removes whitespace between HTML tags.

Example:

```
{% spaceless %}
<div>
    <p>Test test test</p>
</div>
{% endspaceless %}
```

After rendering:

```
<div><p>Test test test</p></div>
```

Note: It will not remove other whitespace.

New in version 0.8.

templatetag

Not implemented, but exists in Zotonic for forward compatibility with future ErlyDTL and Django versions.

translate

Translate a text value using gettext.

Translate the text contained in the tag into the currently selected language.

Example:

```
{_ translate me _}
```

If the active language is "nl" then this will output "vertaal mij". Of course depending on the available translations. When a translation is not available then the text is output as-is without any translation.

See also:

_ (extended translation) (page 440).

url

Generate the URL for a named dispatch rule. In this way it is possible to automatically change the generated URLs when the dispatch rules are modified.

For example to generate the URL to the admin for editing a *page*, use:

```
{% url admin_edit_rsc id=myid %}
```

Assuming myid is 42 then this will generate (on most Zotonic sites) the URL "/admin/edit/42". The name "admin_edit_rsc" can be found in the dispatch-mod_admin-dispatch rules of *mod_admin* (page 275). Which *dispatch rules* (page 12) are available depends on which *Modules* (page 271) are enabled.

When the dispatch rule named in the first argument is unknown then an empty string is returned. There is no error message. This is to prevent breaking the web site when modules are enabled or disabled.

Arguments not named in the path of the dispatch rule are added to the query string of the returned URL:

```
{% url admin_edit_rsc id=42 foo="bar" %}
```

Returns the URL "/admin/edit/42?foo=bar".

Please note that the best way to generate the URL of a page (resource) is to use:

```
{{ m.rsc[myid].page_url }}
```

Generate absolute URLs

By default, the `{% url %}` tag generates relative URLs. Add the argument `use_absolute_url` to generate absolute URLs that include the scheme and hostname:

```
{% url admin_edit_rsc id=42 foo="bar" use_absolute_url %}
```

will return a URL like "http://example.com/admin/edit/42?foo=bar".

Finding out the current dispatch rule

The name of the current dispatch rule is always available in a template under the name `zotonic_dispatch`.

Check *Global variables* (page 22) for a full overview of variables that are always available in the templates.

with

Assign a complex value to a variable.

The `{% with %}` tag assigns the result of a variable expression to a new variable. This is useful when accessing an "expensive" method (e.g., one that hits the database) multiple times. The `{% with %}` tag is often used in conjunction with the search model *m.search* (page 432).

For example:

```
{% with m.search[{latest cat="news"}] as latest_news %}
  The latest {% latest_news|length %} news articles:
  {% for id in latest_news %}
     {{ m.rsc[id].title }}
  {% endfor %}
{% endwith %}
```

This outputs the number of latest news articles and also the titles of the news articles. The search is only done once when the `{% with %}` tag assigns the variable "latest_news".

Multiple assignments

It is also possible to assign multiple variables with a single *with* statement, like this:

```
{% with "value1", "value2" as foo, bar %}
  {{ foo }}
  {{ bar }}
{% endwith %}
```

This will output `value1 value2`.

Custom Tags

Custom tags are internally called *scomp*, from *Screen COMPonents*. Custom tags are implemented by *Modules* (page 39), so availability depends on the activated state of the corresponding module.

Shows statistics for a module/language combination about how many translation strings have been translated into the given language:

```
{% admin_translation_statistics module=`mod_translation` lang=`nl` %}
```

Renders something like:

```
<td class="perc-60"><span>120 / 221</span></td>
```

To indicate that 120 out of 221 strings in *mod_translation* (page 317) have been translated into dutch.

Makes a button with an action attached.

This is an easy way to make a button and attach one or more actions or a postback.

For example:

```
{% button text="Click me" action={alert text="Hello Word!"} %}
```

This show a button with the text "Click me". When clicked it will trigger the action-alert action, showing an alert message with the text "Hello World!".

Another example:

```
{% button text="Postback" postback={my_postback arg1=1 arg2=2} %}
```

When clicked it will call the `event/2` function in the controller that served the page. The function will be called as:

```
event(#postback{message={mypostback, [{arg1,1}, {arg2,2}]},
    trigger=TriggerId, target=TargetId}, Context)
```

Where TriggerId and TargetId are both the HTML id of the button.

`button` accepts the following arguments:

Argument	Description	Example
text	The text on the button, defaults to "Submit"	text="click me"
postback	An event sent to the delegate or the resource serving the page.	postback="hello"
tag	The type of HTML tag that will be created. Defaults to "button".	tag="a"
delegate	The name of the erlang module to be called for handling the postback.	delegate="myresource"
action	The action to be triggered when the button is clicked. There can be more than one action argument.	action={show target="msg"}
id	Id of the button.	id=#submit
class	The css class of the button. This argument can be repeated to add multiple classes.	class="submit"
style	The css style of the button.	style="color: #fc0"
tabindex	The value for the *tabindex* property.	tabindex=1
type	The type attribute of the button.	type="submit"
title	The title attribute of the button.	title="click to submit"
disabled	The disabled attribute of the button, set to true or false. When the button is disabled then the class "disabled" id added to the class list.	disabled=true

Show a pie chart.

This is an utility tag providing a simplified interface to the pie chart feature of the {% google_chart %} tag. It has an easier way to define the data.

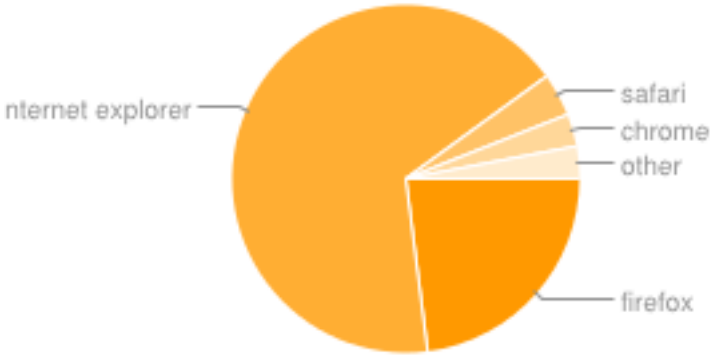
Example of simple pie chart:

```
{% chart_pie data=[["firefox",23],
                   ["internet explorer", 67],
                   ["safari",4],
                   ["chrome",3],
                   ["other", 3]]
%}
```

This generates the following image tag:

```
<img class='google_chart' alt='google chart'
   src='http://chart.apis.google.com/chart?&cht=p&chts=909090,10&chs=300x150&chg=0,
↪0,1,5&chf=bg,s,ffffff|c,s,ffffff&chdlp=b&chbh=-3,3,7&chxt=x&
↪chxl=0:|firefox|internet%20explorer|safari|chrome|other&chxs=0,909090,10&chco=&
↪chds=0,100&chd=t:23,67,4,3,3&chls=1,1,0' width='300' height='150' />
```

Or, as an image:



The tag chart_pie accepts the following arguments:

Argument	Description	Example
data	The data for the pie chart. A list of pairs of {label, value} or [label, value].	[{“nl”,300},{uk,”200”}]
colors	The colors for the pies. A list of colors, when there are more data points than colors then the colors are interpolated. Colors are specified in hexadecimal. Defaults to Google default colors.	colors=[”ffcc00”,”ccff00”,”00ffcc”]
threed	Set to true to have a 3D effect on the pie chart. Defaults to false.	threed=true
width	The width of the generated pie chart, in pixels. Defaults to 300.	width=450
height	The height of the generated pie chart, in pixels. Defaults to 150.	height=200

Other arguments can be found at the scomp-google_chart tag.

See also:

scomp-google_chart and scomp-chart_pie3d.

Show a pie chart with 3D effect.

This scomp is just a convenient interface to the {% chart_pie %} scomp with the `threed` argument set to true.

For example:

```
{% chart_pie3d data=[["firefox",23],
                     ["internet explorer", 67],
                     ["safari",4],
                     ["chrome",3],
                     ["other", 3]]
%}
```

Gives:

```
<img class='google_chart' alt='google chart' src='http://chart.apis.google.com/
↪chart?&cht=p3&chts=909090,10&chs=300x150&chg=0,0,1,5&chf=bg,s,ffffff|c,s,ffffff&
↪chdlp=b&chbh=-3,3,7&chxt=x&chxl=0:|firefox|internet
↪%20explorer|safari|chrome|other&chxs=0,909090,10&chco=&chds=0,100&chd=t:23,67,4,
↪3,3&chls=1,1,0' width='300' height='150'/>
```

Or, as an image:

See also:

scomp-google_chart and scomp-chart_pie.

Shows which variables are assigned for use in the current template's scope:

```
{% debug %}
```

This displays a HTML table with columns for the variable's name and its value, and displays one variable per row:

```
<table>
<tbody>
  <tr>
      <td>session_id</td>
      <td>
          <tt>"qUHoeKodUHXbpwrcl6Vj"</tt>
      </td>
  </tr>

  <tr>
      <td>q</td>
      <td>
          <tt>[{"zotonic_host","examplesite"},{"zotonic_dispatch","home"}]</tt>
      </td>
  </tr>

  <tr>
      <td>template</td>
      <td>
          <tt>"home.tpl"</tt>
      </td>
  </tr>

  <tr>
      <td>zotonic_dispatch</td>
      <td>
          <tt>home</tt>
      </td>
  </tr>
</tbody>
</table>
```

See also:

print (page 454)

Mark a html element as draggable.

The draggable tag is used in conjunction with the {% droppable %} tag to implement drag & drop. Elements that are marked as draggable can be dropped on elements marked as droppable. Drag & drop generates dragdrop events that are sent to the *controller* or the *delegate*.

For example:

```
<div id="drag1">Drag me</div>
{% draggable id="drag1" tag="drag-one" %}
```

Now the div with id “drag1” can be dragged. When it is dropped then `event/2` of the controller or delegate Erlang module is called signaling the drag (and also the drop to the module receiving the droppable events):

```
event({drag, Drag, Drop}, Context).
```

Where both Drag and Drop are `#dragdrop` records:

```
-record(dragdrop, {tag, delegate, id}).
```

The draggable tag accepts the following arguments:

Argument	Description	Example
id	The id of the element that becomes draggable.	id=”drag1”
tag	The tag of the draggable that is sent as part of the drag and drop events. This can be any value, including a tuple.	tag={subject_list id=42 changed=false}
clone	Clone the element when dragging or drag the element itself. Defaults to false.	clone=true
revert	When the element has to revert to its starting position. Defaults to “invalid”, i.e. when the drop was above an invalid position. Other options are true, false and “valid”.	revert=false
axis	Constrain the drag movement to either the x or y direction. Normally the drag is not constrained. Acceptable values are “x” or “y” axis=”x”	
handle	The css selector that is the handle to drag with. Defaults to the whole element.	handle=”handleclass”
group	The name of this drag group, for use in the droppable element’s “accept” argument. Multiple groups are allowed.	group=”edges”
opacity	Change the opacity while dragging. Defaults to “0.8”.	opacity=”0.5”
delegate	The Erlang module that will receive the drag event after a successful drop.	

See also:

the scomp-droppable tag.

Mark an element as valid drag destination.

The droppable tag is used in conjunction with the {% draggable %} tag to implement drag & drop. Elements that are marked as droppable can receive drops of draggable elements. Drag & drop generates dragdrop events that are sent to the *controller* or the *delegate*.

For example:

```
<div id="dropzone">Drop your stuff here</div>
{% droppable id="dropzone" tag="drop-tag" %}
```

Now draggable elements can de dropped onto the div with id “dropzone”. When a draggable is dropped then `event/2` of the controller or delegate Erlang module is called signaling the drop (and also the drag to the module receiving the draggable events):

```
event({drop, Drag, Drop}, Context).
```

Where both Drag and Drop are `#dragdrop` records:

```
-record(dragdrop, {tag, delegate, id}).
```

The droppable tag accepts the following arguments:

Argument	Description	Example
id	The id of the element that will accept drops of draggables.	id="dropzone"
tag	The tag of the droppable that is sent as part of the drag and drop events. This can be any value, including a tuple.	tag={subject id=123}
active	The droppable will have this CSS class added when there is an acceptable draggable being dragged.	active="draghere"
hover	The droppable will have this CSS class added when there is an acceptable draggable hovering over it.	active="dropnow"
accept	The group the droppable accepts. See the group argument of the draggable. A droppable can accept multiple groups, just repeat the accept argument.	accept="edges"
delegate	The Erlang module that will receive the drop event after a successful drop.	

See also:

the scomp-draggable tag.

Make charts with Google.

This tag interfaces to the Google chart API to dynamically generate charts.

For example:

```
{% google_chart type="line" axis={axis position="bottom" labels=["jan","feb","mar
↪"]}
   axis={axis position="left" labels=[0,25,50,75,100]} data=[{data values=[20,80,
↪33]}] %}
```

Will generate the following image tag:

```
<img class='google_chart' alt='google chart'
   src='http://chart.apis.google.com/chart?&amp;cht=lc&amp;chts=909090,10&amp;
↪chs=300x150&amp;chg=0,0,1,5&amp;chf=bg,s,ffffff|c,s,ffffff&amp;chdlp=b&amp;chbh=-
↪3,3,7&amp;chxt=x,y&amp;chxl=0:|jan|feb|mar|1:|0|25|50|75|100&amp;chxs=0,909090,
↪10|1,909090,10&amp;chco=&amp;chds=0,100&amp;chd=t:20,80,33&amp;chls=1,1,0'
   width='300' height='150'/>
```

View the chart.

Google chart accepts the following arguments:

Argument	Description	Example
type	Kind of chart is generated. One of “line”, “sparkline”, “stacked_horizontal_bar”, “stacked_vertical_bar”, “grouped_horizontal_bar”, “grouped_vertical_bar”, “pie” or “pie3d”. Defaults to “line”.	type=”sparkline”
id	The id of the generated image tag.	id=”mychart”
class	CSS class of the generated image tag. The class “google_chart” is always added.	class=”chart”
style	CSS style attribute for the generated image tag.	style=”border: 1px solid #fcc”
title	Title shown on the chart.	title=”Browser shares”
color	Color for the title. Must be in hexadecimal, defaults to “909090”.	color=”ffcc00”
font_size	Font size in pixels for the title. Defaults to 10.	font_size=12
width	Width of the generated chart. Defaults to 300.	width=450
height	Height of the generated chart.	height=200
grid_x	X axis grid step size.	grid_x=10
grid_y	Y axis grid step size.	grid_y=10
grid_line_length	Length of line segment for the grid lines, defaults to 1.	grid_line_length=1
grid_blank_length	Length of gaps in the grid lines, defaults to 5.	grid_blank_length=5
background_color	Background color for the complete chart. in hexadecimal, defaults to “ffffff”.	background_color=”331133”
chart_color	Background color for the chart area. In hexadecimal, defaults to “ffffff”.	chart_color=”113311”
legend_location	Where the legend is placed. One of “top”, “left”, “bottom” or “right”. Defaults to “bottom”.	legend_location=”right”
axis	Description of an axis. You can given more than one axis argument. See the section *Axis styles and labels* (page 463) below.	
data	The data to be shown. You can give more than one data argument. See *Data* (page 464) definition below.	
bar_space	Space in pixels between the bar of a bar chart. Defaults to 3.	bar_space=5
bar_group_space	Space in pixels between the groups of bars of a bar chart. Defaults to 7.	bar_group_space=10

Axis styles and labels

Axis styles and labels are available for line charts and bar charts.

An example for an axis argument is:

```
axis={axis font_size=10 color="909090" position="top" labels=["jan", "feb", "mar"]}
```

This defines an axis that will be displayed above the chart. It has three labels and will be displayed in a 10 pixel font with color “909090”. You can give a multiple axis arguments and also a list of axes for each argument.

Valid arguments for an axis are:

Argument	Description	Example
font_size	Size of the labels in pixels. Defaults to 10.	font_size=12
color	Color for the label texts, in hexadecimal RGB. Defaults to “909090”.	color=”cc0000”
position	Which axis is defined. One of “top”, “right”, “bottom” and “left”. Defaults to “top”. You can have multiple definitions for a position.	position=”bottom”
labels	A list with labels displayed. The labels will be evenly distributed over the axis.	labels=[2006,2007,2008]

Data

All data definitions to be displayed. Each data definition is a record with arguments. You can supply all data sets at once or with multiple data arguments.

Example with one data set:

```
data=[{data line_width=1 min_value=1 max_value=100 values=[10,20,42,34,68,73,80]}]
```

Valid arguments for a data record are:

Argument	Description	Example
line_width	The width of the line in pixels. Defaults to 1.	line_width=2
line_length	Length of line segment in pixels. Defaults to 1.	line_length=1
blank_length	Length of blank segment in pixels. Defaults to 0.	blank_length=1
min_value	The minimum value for the data set, used for the axis. Defaults to 0.	min_value=-100
max_value	The maximum value for the data set, used for the axis. Defaults to 100.	max_value=100
color	The color used for this data set. Hexadecimal RGB value.	color="ffcc00"
legend	Label for the dataset as shown in the legend.	legend="monthly sales"
values	The values for drawing the chart. Must be a list.	values=[0,10,5,8]

See also:

tags scomp-chart_pie and scomp-chart_pie3d.

Render a JS-aided inplace textbox.

Example:

```
{% inplace_textbox value="def.val." delegate="my_resource" hint="edit" %}
```

Todo

Improve documentation

Custom tag which adds a 'loader' image to the page and performs a one-time action when loader comes into view.

`mod_geomap` uses this to load the map JavaScript once the admin widget has been opened by the user.

Example:

```
<div id="{{ #lazy }}">
    {% lazy action={update target=#lazy id=id template="_geomap_admin_location_map.
↪tpl"} %}
</div>
```

`lazy` accepts the following arguments:

Argument	Description	Example
image	The source of the image tag. Defaults to */lib/images/spinner.gif*.	image="/lib/images/loading.gif"
class	The css class of the image. Defaults to *z-lazy*.	class="loader"

Live updating templates connected to MQTT topics.

This scomp renders templates that are automatically rerendered after a publication to a MQTT topic.

Note: This scomp is provided by *mod_mqtt* (page 301), this must be enabled.

Example

An example of a template showing the newest content of a resource:

```
{% live template="_detail.tpl" topic=id id=id %}
```

This renders the template `_detail.tpl`. If the resource with id *id* is updated then the template will be replaced with a freshly rendered template.

The scomp can subscribe to multiple topics at once.

Add the argument `catinclude` to do a *catinclude* instead of a normal *include*. For a *catinclude* the argument *id* must be present.

Live topics

Any MQTT topic can be used. The topics are interpreted as local to the page. There are three special topics:

- Use any integer to map to the resource's update topic. For example if id is *1234* then the topic will be `~site/rsc/1234`
- Use the tuple `{object id=... predicate=...}` to listen to changes of a specific predicate of a page. An example of a mapped topic is `~site/rsc/1234/o/author`
- Use the tuple `{object id=...}` to listen to changes of outgoing connections from a page. An example of a mapped topic is `~site/rsc/1234/o/+`
- Use the tuple `{subject id=... }` to listen to changes of incoming connections to a page. An example of a mapped topic is `~site/rsc/1234/s/author`

Live actions

It is possible to wire actions or postbacks to a MQTT topic.

Use `{% wire type={mqtt topic=... topic=...} %}` to connect to one or more MQTT topics.

Example:

```
{% wire type={mqtt topic="~site/public/hello"} action={growl text="hello"} %}
```

And in Erlang this will trigger the above *growl*:

```
z_mqtt:publish(<<"~site/public/hello">>, <<>>, z_acl:sudo(z:c(mysite))).
```

See also scomp-wire

Inserts a piece of "lorem ipsum" text into the page.

The loremipsum tag inserts the following fake-latin text:

```
```

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam lobortis, lacus id molestie suscipit, enim leo pharetra velit, ac cursus felis mauris at velit. Cras ultrices, massa pharetra faucibus cursus, neque leo bibendum mauris, eu vulputate leo magna vitae lacus. Etiam pharetra gravida elementum. Maecenas lacinia, sem sed eleifend euismod, risus velit interdum nulla, convallis sagittis orci dui a metus. Aliquam tristique orci a ipsum dapibus viverra. Donec vestibulum varius ante, vitae placerat magna ultrices ac. Donec nec ante magna. Donec porttitor, arcu a condimentum aliquam, lectus nunc rhoncus quam, id ornare neque ligula quis dui. Pellentesque sit amet lectus augue, ut ullamcorper nisi. Donec et rutrum sem. In porta ultricies nibh, in sagittis lacus euismod at. Etiam consectetur tristique tellus, quis tristique dui vulputate vel. Curabitur sagittis gravida dui, vel sagittis lectus suscipit ac.

The loremipsum tag accepts the following arguments:

Argument	Description	Example
words	The number of words to be displayed from the text above.	words=10

New in version 0.5.

Changed in version 0.6: Added `words` argument.

Show the mailinglist subscription form to subscribe to a certain mailinglist id.

Parameters:

`id` Required; the id of the mailinglist *resource* that is being subscribed to.

`template` Which form template to render. Defaults to the template `_scomp_mailinglist_subscribe.tpl`.

All other parameters are passed in to the template which is being rendered.

The form is at least supposed to have an *email* input field. Besides *email*, it can have *name_first*, *name_surname_prefix* and *name_surname* fields, which will be stored in the recipient table.

Show a page menu.

This tag is part of the module mod_menu. The `{% menu %}` tag is used to generate the HTML for the menu defined in the admin.

You can define multiple menus in your site. By default there is one menu, called "main_menu". If you want another one, create a page of type "page menu" (under "Categorization") and start editing your menu. You can use the "menu_id" argument to select which menu you want to display.

Example:

```
{% menu id=id %}
```

Generates something like:

```
<ul id="navigation" class="nav">
  <li>
    <a href="/" class="welcome">Home</a>
  </li>
  <li>
    <a href="/features" class="page_features">Features</a>
  </li>
  <li>
    <a href="/documentation" class="documentation active">Documentation</a>
  </li>
  <li>
    <a href="/documentation/628/installation" class="page_install">Install</a>
  </li>
</ul>
```

The menu has the following features:

- The menu is a unordered list.
- The id of the menu is `navigation` and can be prepended with param `id_prefix`.
- The class of the menu is set with param `class` (default `nav`).
- Menu items are a <li> with a single <a>
- The link of the menu item referring to the current page has the class `active`.
- Every link also gets the unique name of the target as a class.
- Every menu item can have single level submenus. A submenu has the same properties as the menu.

Argument	Description	Example
id	Set this to the id of the current shown page and it wil highlight its page path.	
menu_id	The id of the menu that you want to display. If left empty, the main menu is shown.	
id_prefix	String prepended to menu id.	
class	HTML class for the list; default "nav".	
maxdepth	Maximum depth of the menu; default 999.	

Show a pager for search results.

This generates a pager as seen on the search results pages. It is used in conjunction with a paged search result.

For example, a fulltext search where the search parameters come from the query string:

```
{% with m.search.paged[{fulltext cat=q.qcat text=q.qs page=q.page}] as result %}
  <ul>
    {% pager result=result dispatch="admin_overview_rsc" qargs %}
    {% for id,score in result %}
      <li><a href="">{{ m.rsc[id].title }}</a></li>
    {% empty %}
      <li>Nothing found</li>
    {% endfor %}
  </ul>
{% endwith %}
```

This will show a list of titles and above that the links to the next, previous and other pages.

Note: that we are using `m.search.paged` here and not *m.search* (page 432). The pager only works with results from `m.search.paged`.

The generated pager code will look something like (when searching for the text "filter"):

```
<ul class="pager block">
<li>« prev</li>
<li class="current"><a href="/admin/overview?qs=filter&page=1">1</a></li>
<li><a href="/admin/overview?qs=filter&page=2">2</a></li>
<li><a href="/admin/overview?qs=filter&page=3">3</a></li>
<li><a href="/admin/overview?qs=filter&page=4">4</a></li>
<li class="pager-sep">...</li>
<li><a href="/admin/overview?qs=filter&page=5">5</a></li>
<li><a href="/admin/overview?qs=filter&page=2">next»</a></li>
</ul>
```

The pager tag accepts the following arguments:

Argument	Description	Example
result	The result from a search. This must be a `#search_result` or `#m_search_result` record. Note that this must be the result of a `m.search.paged` and not of a `m.search` call.	result=mysearchresult
dispatch	Name of the dispatch rule to be used for the page urls. Defaults to the dispatch rule of the current page.	dispatch="searchresult"
qargs	Append all the arguments in the HTTP request's query string whose name starts with a 'q' as an argument to the dispatch rule.	qargs
hide_single_page	When this argument is true, do not show the pager when the result fits on one page (e.g. the pager will be useless).	hide_single_page=1
*	Any other argument is used as an argument for the dispatch rule.	

Show a given survey (with the `id` parameter) as a "poll". This presents a simpler interface, in which the user is directly asked to enter some information, e.g. make a choice between certain things:

```
{% poll id=123 %}
```

This tag is the placeholder where all generated JavaScript scripts will be output on the page.

Zotonic generates JavaScript for the actions and other template logic. This script needs to be added to the page. The `{% script %}` scomp designates the place where the `<script>` element with all the generated javascript can be placed.

Normally the `{% script %}` scomp is placed at the end of the page, just above the `</body>`.

Note that all JavaScripts generated after the `{% script %}` scomp will not be included in the generated page. Only a single `{% script %}` scomp is allowed on any page.

Example:

```
    {# at the bottom of the template ... #}
    {% script %}
  </body>
</html>
```

This will generate something similar to:

```
<script type='text/javascript'>
$(function() {
z_pageid="MouKz4PgrcROM5efU8iL";
z_postback_loop();

$('#vtretq').bind('click', function(event) { window.location = "/admin/edit/647";␣
↪return z_opt_cancel(this); } );
z_init_postback_forms();
z_default_form_postback = "bVcYISt9JOG/AQZkP9SOZmc//
↪GqDaAVrAAZzdWJtaXRkAAl1bmRlZmluZWRkAAl1bmRlZmluZWRqZAANcmVzb3VyY2VfcGFnZQ==";
});
</script>
  </body>
</html>
```

Note that the contents of this block will be completely different per page.

The script scomp can have the following arguments:

Argument	Description	Example
nostartup	Exclude the page initialization code from the script, only includes the scripts from actions etc. Default is to include the page initialization code.	nostartup
nostream	Do not start the bi-directional communication layer (over WebSockets or comet).	nostream
format	Select a different format than the `<script/>` tag. For now this accepts `"html"` (for the `<script/>` tag), `"escapejs"` for an escaped javascript string, and `"js"` for a normal javascript string. Default is `"html"`.	format=”html”

WebSockets / Comet communication

Unless `nostream` is added as a parameter, this tag also causes the WebSockets or Comet communication layer to be initiated.

When available, a WebSocket connections is opened, otherwise a long polling Comet connection is started. The WebSockets connection will also be used for sending Ajax requests to the server.

See also *Transport* (page 49) for details on the bi-directional communication.

Mark an element as sortable.

Sortables are part of a sorter. Sortables in a sorter can be reordered by drag & drop.

Example:

```
{% sorter id="sorter" tag="mysorter" %}
{% sortable id="sort1" tag=1 %}
{% sortable id="sort2" tag=2 %}
<ul id="sorter">
  <li id="sort1">Sortable 1</li>
  <li id="sort2">Sortable 2</li>
</ul>
```

This creates a list where the order of the list items can be changed by dragging and dropping them.

When the order of the sortables is changed, an event is sent to the sorter's page *controller* or *delegate*. The event contains the ordered list of sortable tags.

The event handler function is called like:

```
event({sort, Sortables, Sorter}, Context).
```

Where "Sortables" is the list of sortables and "Sorter" is the sorter. Both are "#dragdrop" records:

```
-record(dragdrop, {tag, delegate, id}).
```

Where "tag" is the tag of the sortable or sorter, "delegate" is the module that handles the event and "id" is the HTML id of the sortable or sorter.

Note: that when the tag is a string then the `#dragdrop` tag will be an atom.

The sortable can have the following arguments:

Argument	Description	Example
id	The HTML id of the sortable element.	id="mysortable1"
tag	Tag that identifies the sortable to the event handler. Can be any value. A string will be converted to an atom.	tag="my_atom_value"
delegate	The delegate of the sortable, currently unused will be passed in the sortables's #dragdrop record.	delegate="mymodule"
class	A CSS class that will be added to the sortable. The class "sortable" will always be added.	class="dragitem"

See also:

the scomp-sorter tag.

A sorter is a container for sortables.

A sorter contains sortables and handles the events when the order of the sortables is changed. Sortables in a sorter can be reordered by drag & drop.

Example:

```
{% sorter id="sorter" tag="mysorter" %}
{% sortable id="sort1" tag=1 %}
{% sortable id="sort2" tag=2 %}
<ul id="sorter">
  <li id="sort1">Sortable 1</li>
  <li id="sort2">Sortable 2</li>
</ul>
```

This creates a list where the order of the list items can be changed by dragging and dropping them.

When the order of the sortables is changed, an event is send to the sorter's page *controller* or *delegate*. The event contains the ordered list of sortable tags.

The event handler function is called like:

```
event({sort, Sortables, Sorter}, Context).
```

Where "Sortables" is the list of sortables and "Sorter" is the sorter. Both are `#dragdrop` records:

```
-record(dragdrop, {tag, delegate, id}).
```

Where "tag" is the tag of the sortable or sorter, "delegate" is the module that handles the event and "id" is the HTML id of the sortable or sorter.

Note: that when the tag is a string then the `#dragdrop` tag will be an atom.

The sorter can have the following arguments:

Argument	Description	Example
id	The HTML id of the sortable element.	id="mysorter"
tag	Tag that identifies the sortable to the event handler. Can be any value. A string will be converted to an atom.	tag="my_atom_value"
delegate	The delegate of the sorter. The sort event will be send to the delegate module. Defaults to the resource that handled the page request.	delegate="mymodule"
class	A CSS class that will be added to the sorter. The class "sorter" will always be added.	class="bunny-sorter"
handle	jQuery selector for the handles of the sortables. When not defined then the whole sortable can be clicked on for dragging.	handle=".sortable-handle"
connect_group	Name of the group this sorter connects with. Sortables from this sorter can then be dragged to sorters with that group name. This argument can be repeated to connect with multiple groups. Special values are "all" and "none" to either connect to all other sorters or to no other sorter.	connect_group="bunnies"
group	The group of this sorter. Used in connection with the "connect_group" argument. Sortables can be dragged between sorters of the same group.	group="cows"
axis	If defined the items can only be dragged horizontally or vertically. Possible values are "x" and "y".	axis="y"
containment	Constrains dragging of the sortables to within the bounds of the specified element. Possible values are "parent", "document", "window", or a jQuery selector.	containment="parent"
opacity	Opacity a sortable is set to when being dragged. Defaults to "1.0".	opacity="0.8"
placeholder	Class that gets applied to the otherwise white space that will show between sortables as the new place of the sortable.	class="drophere"

See also:

the scomp-sortable tag.

Add an AJAX activity indicator.

Whenever an AJAX call is made the HTML element with the id `#spinner` will be shown during the call. It will not be shown during *Comet* activity.

You can add a spinner element to your page yourself or use this tag to place it somewhere on your page.

Example:

```
{% spinner %}
```

Outputs the HTML code:

```
<div id="spinner" class="spinner" style="display: none">
  <img alt="activity indicator" src="/lib/images/spinner.gif" />
</div>
```

The spinner tag accepts a single argument "image". The "image" argument must contain the URL for the image displayed. It defaults to "/lib/images/spinner.gif".

This tag is deprecated. WebSocket / Comet streams are started automatically when the scomp-script tag is used.

See also:

scomp-script

Returns a html fragment that can be used as an example for a survey question. The `type` parameter determines which survey question type to render:

```
{% survey_example type="likert" %}
```

Mainly used in the admin interface when editing surveys.

Make a HTML element into a tab set.

This is a simple interface to the jQuery UI tabs functionality. It will show tabs to switch between different panes.

The only argument is "id" which is the id of the container of the tabs.

The following example will show an interface with three tabs:

```
{% tabs id="tabs" %}
<div id="tabs">
  <ul>
    <li><a href="#tabs-1">Nunc tincidunt</a></li>
    <li><a href="#tabs-2">Proin dolor</a></li>
    <li><a href="#tabs-3">Aenean lacinia</a></li>
  </ul>
  <div id="tabs-1">
    <p>Proin elit arcu, rutrum commodo, vehicula tempus.</p>
  </div>
  <div id="tabs-2">
    <p>Morbi tincidunt, dui sit amet facilisis feugiat.</p>
  </div>
  <div id="tabs-3">
    <p>Mauris eleifend est et turpis.</p>
  </div>
</div>
```

Note: There is no default styling for jQuery UI elements in the zotonic CSS files. See these two threads in the zotonic users mailinglist: does tabs scomp still work?, and playing with tabs scomp. See also the jQuery UI documentation.

The validator tag accepts the following arguments:

Argument	Description	Example
id	The id of the input element to be validated.	id="password_field"
type	The validator for the input element. Can be specified multiple times to add multiple validators. The value depends on the validator choosen but is always a record `{validatorname arg=value ...}`	type={acceptance}
trigger	Id of the element that triggers the validation. Defaults to the value of "id". The validator is triggered by a change of this. Almost never used.	trigger="title"
target	Target of validator, defaults to the trigger. Almost never used.	
name	Use this when the name of the input element is unequal to the id. The name is used by the server side code to validate the received input. Defaults to the target argument (which defaults to id).	name="password_field"
valid_message	Message to show when the field passes validation. Defaults to the empty string.	valid_message="ok!"
failure_message	Argument passed to the type argument. See individual validators.	
message_after	The id of the element after which the failure message should be shown. Defaults to the id argument.	message_after="signup_tos_agree"
only_on_blur	Normally validates on change, unles only_on_blur is set.	only_on_blur
wait	Time in msec to wait for validation after the last keystroke. Default: 0.	wait=100
only_on_submit	Whether the validation should be done when entering data or only on submit of the form. Set this to suppress validation when entering data.	only_on_submit

See also:

- the list of *Validators* (page 476)
- *Forms and validation* (page 27) in the Developer Guide

Connect actions and events to a HTML element.

The wire tag is the basis for most Ajax interaction on web pages. It allows to connect actions to HTML elements. Examples of *Actions* (page 320) are action-show / action-hide elements or action-postback to the server.

Wire actions to an element

The primary use of wire is to connect an action to a HTML element or javascript event.

Example:

```
{% wire id="show" action={show target="message"} %}
<a id="show" href="#">Click to show a message</a>
<p style="display: none" id="message">Hello World!</p>
```

A click on the link will show the message "Hello World!".

Wire postbacks to the server

Wire can post an event back to the server. Use for this the "postback" argument:

```
{% wire id="mybutton" postback={hello world="round"} %}
<button id="mybutton">Post event to server</button>
```

When the button is clicked the event function of the page controller will be called as:

```
event({postback, Postback, TriggerId, TargetId}, Context).
```

Where "Postback" will be `{hello, [{world,"round"}]}` and both "TriggerId" and "TargetId" will be "mybutton".

Wire form submit events

Wire is also used to connect a form to the event routine handling the form.

Example:

```
{% wire id="myform" type="submit" postback="some_tag" action={toggle target=
↪"message"} %}
<form id="myform" method="post" action="postback">
  <input type="text" name="title" value="" />
  <button id="mybutton" type="submit">Submit</button>
</form>
```

The wire tag redirects the submit of the form to the event routine of the controller. A submit will also toggle the visibility of the "message" element on the page. Note that the action is "postback", this is obligatory.

The event routine will be called as:

```
event({submit, Tag, FormId, TargetId}, Context).
```

Where Tag will be the postback set by the wire tag (in the example the atom `some_tag`) and FormId and TargetId are both the HTML id of the submitted form. The posted input fields can be fetched using `z_context:get_q/2`, `z_context:get_q_all/1` or `z_context:get_q_validated/2`.

```
Title = z_context:get_q("title", Context),
AllArgs = z_context:get_q_all(Context);
```

There are some extra arguments added to every form post:

Post argument	Description	Example
z_trigger_id	Id of the HTML element that triggered the submit.	"mybutton"
z_pageid	Id of the page in the browser, used to connect comet and other communictation between the browser and the server.	"1uTs-bzIsWqmdPmNopF32"
postback	Signed postback set by the wire tag. Handled internally.	

Wire to the page load or unload

A `{% wire %}` without an id will bind the actions and/or postback to the window instead of an element. Omitting a type as well will execute all actions and/or postback on page load.

Example:

```
{% wire action={alert text="Welcome to this page."} %}
```

The type "unload" will execute all actions and/or postback when leaving the page:

```
{% wire type="unload" action={alert text="Bye."} %}
```

Call a wire action from JavaScript

Use `{% wire name="myname" %}` to define a named action and trigger it from JavaScript with `z_event("myname")`. See: *Wired events* (page 48).

Wire an action to a MQTT topic

Use `{% wire type={mqtt topic=... topic=...} %}` to connect to one or more MQTT topics.

Example:

```
{% wire type={mqtt topic="~site/public/hello"} action={growl text="hello"} %}
```

And in Erlang this will trigger the above *growl*:

```
z_mqtt:publish(<<"~site/public/hello">>, <<>>, z_acl:sudo(z:c(mysite))).
```

Note: *mod_mqtt* (page 301) must be enabled before wiring to a topic

See also scomp-live

Arguments

The wire tag accepts the following arguments:

Argument	Description	Example
id	HTML id of the element the action gets connected to. When the id is not given then the event is bound to the window.	id=”mybutton”
type	The type of the event triggering the action. Defaults to “click”. Other types are: “enterkey”, “interval”, “continuation”, “submit” or one of the jQuery events “blur”, “focus”, “load”, “resize”, “scroll”, “unload”, “beforeunload”, “click”, “dblclick”, “mousedown”, “mouseup”, “mousemove”, “mouseover”, “mouseout”, “mouseenter”, “mouseleave”, “change”, “select”, “keydown”, “keypress”, “keyup” or “error”. The types can be extended by modules using the `#action_event_type` notification. The type must be a tuple, an example is the `{mqtt topic=...}` type provided by *mod_mqtt* (page 301)	type=”submit”
propagate	Specify this when you don’t want the event to be canceled after handling the wire. Useful for event types like focus, click etc. .. versionadded:: 0.6.1	propagate
target	Possible target for the action. The meaning of this argument depends on the action, defaults to id.	
action	Action wired to the element. This parameter can be repeated to wire more than one action at a time. The value is a single or a list of action records.	action={toggle target=”message”}
postback	Postback that will be sent to the event handler of the controller or the delegate. Either a string, which will be send as an atom, or a tagged property list. The example will be in Erlang `{myevent, [{foo, 1},{bar,2}]}`.	postback=”ajaxevent” postback={myevent foo=1 bar=2}
delegate	Name of the Erlang module that will receive the postback. Defaults to the controller that handled the page request.	delegate=”event_handler”

See also:

the tag scomp-wire_args and the list of predefined *Actions* (page 320).

Add extra arguments to wired actions.

The tag `wire_args` is used to append extra arguments to actions before wiring those actions. This is useful when an action is handed down as an argument to a template but still needs the id of a local element.

For example:

```
{% with {my_action some_arg=some_value} as my_action %}
  {% for n in [1,2,3,4] %}
    <a id="{{#id.n}}" href="#">click {{n}}</a>
    {% wire_args action=my_action the_link_id=#id.n %}
  {% endfor %}
{% endwith %}
```

This wires the action `{my_action some_arg=some_value the_link_id=...}` to the links.

The following arguments are part of the wire tag and can't be used for argument appending: "id", "type", "target", "action", "postback" and "delegate".

See also:

the scomp-wire tag.

Validators

Validator types for the tag `{% validate %}`.

Check if an input value evaluates to true.

Can be used in combination with a check box that must be checked on submit.

For example:

```
<input type="checkbox" id="accept" name="accept" value="1" />
{% validate id="accept" type={acceptance} %}
```

Arguments:

Argument	Description	Example
failure_message	Message to be shown when the input is true. Defaults to "Must be accepted."	failure_message="Please agree to our TOS."

Check if two inputs are the same.

Useful when a form requires double entry of a password or e-mail address to prevent typos.

Accepts an additional parameter *name* with the name of the other input field.

For example:

```
<input type="password" id="password" name="password" value="" />
<input type="password" id="password2" name="password2" value="" />
{% validate id="password" type={confirmation match="password2"} %}
```

Arguments:

Argument	Description	Example
match	The id of the input field that should have the same value.	match="field1"
failure_message	Message to be shown when the two fields are unequal. Defaults to "Does not match."	failure_message="Please retry."

Support for custom client-side (JavaScript-based) validators.

This call simply adds a `z_add_validator()` JavaScript call which adds a custom validator based on the arguments given in the validator.

Example:

```
<input type="text" name="foobar" id="foobar" value="" />
{% validate id="foobar" type={custom against="window.validate_foobar"} %}
```

And then `validate_foobar` is defined as follows:

```
function validate_foobar(value, args, isSubmit, submitTrigger)
{
    // ... some logic to check the value
    return true or false;
}
```

The `args` are available if the validation is added using the LiveValidation JavaScript API.

`isSubmit` is `true` if the validation is triggered by a form submit, if it was triggered by change or focus events then it is `false`.

`submitTrigger` is the DOM tree node triggering the possible form submit.

Note that this validation does not do any server side validation. Because there is no server side validation, the value of the `input` element is not available via `z_context:get_q_validated/2` but only via `z_context:get_q/2`.

New in version 0.8.

Validate input date against a given date format.

A quick guide to validating a date in Zotonic:

```
<input type="text" id="my_date" name="my_date" value="" />
{% validate id="my_date" type={date separator="-" format="b"} %}
```

This code validates when the user enters a date in the following format:

```
yyyy-mm-dd
```

Arguments

Argument	Description	Example
separator	Character used to separate date parts, such as `/ - \`. Defaults to `"-"`.	`separator="-"`
format	Date format, big endian (starting with year), little endian (starting with day) or middle endian (starting with month). Defaults to `"l"` (little).	`format="m"`

Check if the content of the input field is an e-mail address.

For example:

```
<input type="text" id="email" name="email" value="" />
{% validate id="email" type={email} %}
```

Arguments:

Argument	Description	Example
failure_message	Message to show when the entered value is not an e-mail address. Defaults to “Incorrect E-mail”	failure_message=”Please enter your e-mail address.”

Check if an entered e-mail address is unique, by looking in the *m_identity* (page 420) table for the *email* key:

```
<input type="text" id="email" name="email" value="" />
{% validate id="email" type={email} type={email_unique} %}
```

Optionally, an *rsc_id* parameter can be given to the validator to skip that particular id when doing the uniqueness check. This is useful when you are displaying a form in which the user is editing his own email address.

See also:

m_identity (page 420), validator-username_unique

Regular expression test.

Checks if an input element's value matches a regular expression. There is an optional *negate* argument to validate only when the regular expression does not match.

For example, to test a dutch postal code:

```
<input type="text" id="postcode" name="postcode" value="" />
{% validate id="postcode" type={format pattern="^[0-9][0-9][0-9][0-9] +[A-Za-z][A-
↪Z][a-z]$"} %}
```

Another example, no digits allowed:

```
<input type="text" id="nodigit" name="nodigit" value="" />
{% validate id="nodigit" type={format pattern="[0-9]" negate} %}
```

Accepts the following arguments:

Argument	Description	Example
pattern	The regular expression to match against.	pattern="[0-9][a-z]+"
negate	Specify negate when you want to accept values that do not match the pattern.	negate
failure_message	Message to show when the input value does not match the pattern (or does match the pattern when the negate argument is given). Defaults to "Not valid."	failure_message="Invalid postcode"

Check the length of a text input.

Test if the length of the input's value is more than a minimum and/or less than a maximum length.

Arguments are *minimum* and *maximum*. You can give either or both arguments.

For example, a summary that is neither too long nor too short:

```
<textarea id="summary" name="summary"></textarea>
{% validate id="summary" type={length minimum=20 maximum=200} %}
```

Arguments:

Argument	Description	Example
is	Use when the value must be a specific length.	is=4
minimum	The minimum length of the value.	minimum=4
maximum	The maximum length of the value.	maximum=10
wrong_length_message	Message for when the length is unequal to the value of the "is" argument. Defaults to "Must be . characters long."	
too_short_message	Message for when there are not enough characters entered. Defaults to "Must not be less than . characters long."	
too_long_message	Message for when there are too many characters entered. Defaults to "Must not be more than . characters long."	

A *validator* (page 27) to check whether a resource's name is unique:

```
<input type="text" id="name" name="name" value="" />
{% validate id="name" type={name_unique} %}
```

Optionally, pass an `id` parameter to exclude that particular id when testing for uniqueness. This is useful when you want to exclude the name of the resource currently being edited:

```
<input type="text" id="name" name="name" value="" />
{% validate id="name" type={name_unique id=id} %}
```

You can also pass a `failure_message`:

```
<input type="text" id="name" name="name" value="" />
{% validate id="name" type={name_unique id=id failure_message=_"Eek! Already used!
↪"} %}
```

See also:

Forms and validation (page 27), validator-username_unique

Numerical input and range check.

Checks if the input is a number and within a certain range or equal to a fixed value. At the moment only integer inputs are allowed.

Arguments are *is*, *minimum* and *maximum*.

For example, when the input must be *42*:

```
<input type="text" id="number" name="number" value="" />
{% validate id="number" type={numericality is=42} %}
```

And for a number within a certain range:

```
<input type="text" id="percent" name="percent" value="" />
{% validate id="percent" type={numericality minimum=0 maximum=100} %}
```

Arguments are:

Argument	Description	Example
is	Tests for equality.	is=42
minimum	Minimum value.	minimum=1
maximum	Maximum value.	maximum=100
is_float	Boolean flag which tells if the input can be a floating point number. Defaults to false.	is_float is_float=‘true‘
not_a_number_message	Message to show when the entered value is not a number. Defaults to “Must be a number.”	not_a_number_message=”*”
not_an_integer_message	Message to show when the entered number is not an integer. Defaults to “Must be an integer.”	
wrong_number_message	Message to show when the entered number is unequal to the .is. argument. Defaults to “Must be ..”	
too_low_message	Message for when the entered number is less than the minimum allowed. Defaults to “Must not be less than ..”	
too_high_message	Message for when the entered number is greater than the maximum allowed. Defaults to “Must not be more than ..”	

Performs a custom server side validation of an input value. This allows you to add your own validation logic to HTML form fields.

Start by adding a validator of the type `postback` to your form field in your template:

```
<input type="text" id="username" name="username" value="" />
{% validate id="username" type={postback event="validate_username"} %}
```

The `event` argument declares the name of the event that will be *notified* (page 46). *Handle this event* (page 47) in your site or module:

Listing 5.1: sites/yoursite/yoursite.erl

```
-export([
    observe_validate_username/2
]).

%% The event name passed in your template as event="validate_username",
%% prefixed with observe_
observe_validate_username({validate_username, {postback, Id, Value, _Args}},␣
↪Context) ->
    case is_valid(Value) of
        true ->
            {{ok, Value}, Context};
        false ->
            %% The validation message will be shown in the form
            {{error, Id, "Sorry, that's not valid. Try again!"}, Context}
    end.

%% Some function that returns true or false depending on the validity of the
%% input value
is_valid(Value) ->
    %% ...
```

See also:

- *Forms and validation* (page 27)

Check if an input has been filled in or checked.

For example when a title must be entered:

```
<input type="text" id="title" name="title" value="" />
{% validate id="title" type={presence} %}
```

Extra arguments:

Argument	Description	Example
failure_message	Message to be shown when field is empty. Defaults to "*"	failure_message="Please enter."

Check if an entered username is unique, by looking in the *m_identity* (page 420) table for the given username:

```
<input type="text" id="username" name="username" value="" />
{% validate id="username" type={username_unique} %}
```

Optionally, an `rsc_id` parameter can be given to the validator to skip that particular id when doing the uniqueness check. This is useful when you are displaying a form in which the user is editing his own user name.

See also:

m_identity (page 420), validator-email_unique

See also:

the scomp-validate scomp.

Directory structure

Zotonic's directory structure is somewhat different from a regular Erlang application:

```
zotonic/
  deps/
  doc/
  ebin/
```

```
include/
modules/
priv/
    custom_tags/
    log/
    sites/
src/
    behaviours/
    dbdrivers/
        postgresql/
    erlydtl/
    i18n/
    install/
    models/
    support/
    tests/
user/
    log/
    sites/
        ...
    modules/
        ...
```

`zotonic/`

The main directory contains the startup script "start.sh" and the makefile "Makefile" to build the system.

`zotonic/deps/`

The OTP applications Zotonic depends on. Currently contains Webmachine, MochiWeb and erlang-oauth.

`zotonic/doc/`

Some very simple documentation and installation guide. The main documentation is on this site. Also contains installation instructions for using Zotonic with nginx and an example Varnish configuration file.

`zotonic/ebin/`

All compiled erlang beam files are placed here.

`zotonic/include/`

The main zotonic include file "zotonic.hrl" and basic Webmachine resources used as a basis by other resource modules.

`zotonic/modules/`

All core modules of Zotonic. See the modules documentation for more information.

`zotonic/priv/`

The priv directory is the place where all non core files are placed.

`zotonic/priv/custom_tags/`

Custom tags for the ErlyDTL templates.

`zotonic/priv/log/`

Here all the http access logs are written, there is currently no split between different sites.

`zotonic/priv/sites/`

The directory containing sites which are internal to Zotonic. These are the `zotonic_status` site (see *The Status site* (page 494)), and the `testsandbox` and `testsandboxdb` sites for running the unit tests.

`zotonic/src/`

The Erlang source for the core Zotonic system.

`zotonic/src/behaviours/`

Currently there are two behaviours defined. One for scomps and one for models.

`zotonic/src/dbdrivers/`

The database drivers. Currently, only PostgreSQL is supported as DBMS. As the modules and models sometimes use database specific queries (especially for full text search) it will not be easy to substitute an alternative database. Read this article why Zotonic only supports PostgreSQL.

`zotonic/src/erlydtl/`

The source of the template compiler and runtime files. This is an adapted and extended version of ErlyDTL, which is an Erlang adaptation of the Django Template Language.

`zotonic/src/i18n/`

Internationalization support. Currently, Zotonic supports basic multi-lingual websites. Full support for Internationalization is planned for the 1.0 release.

`zotonic/src/install/`

The database model and basic installation for sites. The files here are used when an empty database is found and needs to be installed.

`zotonic/src/models/`

The data model access files. Implements internal APIs for the different data models for use in Erlang modules and templates. Examples of datamodels are m_rsc, m_config and m_category.

`zotonic/src/support/`

All base Zotonic source code. Here you will find the source code for site supervisors, module supervisors, image resize server, context routines, and much more.

`zotonic/src/tests/`

Contains the EUnit tests for Zotonic.

`zotonic/user`

This directory contains user-modifiable source code which runs in Zotonic, namely user-defined sites and modules.

`zotonic/user/sites/`

A single Zotonic installation is capable of "virtual hosting" serving multiple sites. This directory holds the sites which are created and maintained by you, the users of Zotonic.

This directory is the default location of the `user_sites_dir` configuration variable. See *Global configuration* (page 490).

`zotonic/user/modules/`

This directory holds modules which are not part of the core Zotonic modules, but which are also not site-specific. All modules installed with the `zotonic module install ...` command are placed here.

This directory is the default location of the `user_modules_dir` configuration variable. See *Global configuration* (page 490).

Icons

Including Zotonic icons CSS

Add the CSS file to your template:

```
{% lib
    "css/z.icons.css"
%}
```

This ensures that whenever a Zotonic icon class is used, the image will show up.

Writing Zotonic icons CSS classes

In your template HTML, use this syntax:

```
z-icon z-icon-<name>
```

For instance:

```
<span class="z-icon z-icon-off"></span>
```

Always include `z-icon`, except for the convenience class `zotonic-logo`.

If you want to provide a text label for accessibility (but visually hidden), put the text inside `<em>` or `<span>`:

```
<span class="z-icon z-icon-off"><em>Log off</em></span>
```

Available classes:

```
.z-icon-cross
.z-icon-cross-circle
.z-icon-drag
.z-icon-edit
.z-icon-facebook
.z-icon-google-plus
.z-icon-help
.z-icon-help-circle
.z-icon-info-circle
.z-icon-instagram
.z-icon-linkedin
.z-icon-logo-z
.z-icon-logo
.z-icon-minus
.z-icon-minus-circle
.z-icon-off
.z-icon-ok
.z-icon-ok-circle
.z-icon-plus
.z-icon-plus-circle
.z-icon-twitter
.z-icon-user
.zotonic-logo
```

Buttons

Icons-as-buttons: 2 convenience classes for commonly used buttons are:

```
.z-btn-remove
.z-btn-help
```

These buttons are styled as round buttons with size 16px. Button text inside `<em>` or `<span>` is hidden.

To have consistent close buttons, Bootstrap close buttons `a.close` and `button.close` are overridden by the `z-btn-remove` style.

Usage:

```
<a class="z-btn-remove"><em>Remove</em></a>
```

Social login buttons

Social login buttons are created with `z-btn-social` and show up with a brand icon at the side. Usage:

```
<a href="#" class="btn z-btn-social"><span class="z-icon z-icon-facebook"></span>␣
↪Login with Facebook</a>
```

NOTE: the full syntax that opens a new login window:

```
<a href="{% url logon_service service='facebook' is_connect=is_connect %}"
   class="btn z-btn-social do_popupwindow"
   data-popupwindow="height:300"
   style="color: white; background-color: #44609d">
      <span class="z-icon z-icon-facebook"></span>
      {% if is_connect %}
         {_ Connect with Facebook _}
      {% else %}
         {_ Login with Facebook _}
      {% endif %}
</a>
```

"popupwindow" must be included in the template:

```
{% lib
    "js/modules/z.popupwindow.js"
%}
```

Writing LESS

If you are writing frontend styles in LESS, Zotonic icons can be extended using mixins (found in `extend.less`).

NOTE: The less files have a dependency with `mod_artwork/lib/font-awesome-4`, so you need to include the path to its LESS folder when using the `lessc` command. For example:

```
lessc --include-path="../../../mod_artwork/lib/font-awesome-4/less"
my_input.less my_output.css
```

Or for easier access, create a symlink to the *font-awesome-4* LESS folder and add the symlink to the include-path.

Extending Zotonic icons

To extend a class with a Zotonic icon class, write:

```
.extend_icon(z, @name)
```

For example:

```
.my-btn-help {
   .extend_icon(z, icon-help-circle);
}
```

This will generate the following CSS:

```
(lots-of-classes),
.my-btn-help:before {
    speak: none;
    font-style: normal;
    font-weight: normal;
    font-variant: normal;
    text-transform: none;
    line-height: 1;
    display: inline-block;
    -webkit-font-smoothing: antialiased;
    -moz-osx-font-smoothing: grayscale;
}
(lots-of-classes),
.my-btn-help:before {
    font-family: "Zotonic";
}
.my-btn-help:before {
    content: "\e600";
}
```

The `:before` pseudo-class can be extended to further style the icon. For instance to add a plus icon to a link:

```
.my-plus-link {
    position: relative;
    padding-left: 16px;
    .extend_icon(z, icon-plus);
    &:before {
        position: absolute;
        top: 4px;
        left: 0;
        width: 16px;
        font-size: 13px;
    }
}
```

Extending Material Design icons

1. Enable module mod_artwork.
2. In LESS, add parameter 'md' to the *extend* mixin and pass the character code:

```
.btn-bookmark {
    .extend_icon(md, "\f019");
}
```

The icon (variable) characters can be found in the icons cheatsheet.

Extending FontAwesome 4 icons

1. Enable module mod_artwork.
2. In LESS, add parameter 'fa' to the *extend* mixin:

```
.btn-bookmark {
    .extend_icon(fa, fa-var-bookmark);
}
```

The icon (variable) names can be found in `mod_artwork/lib/font-awesome-4/less/variables.less`.

Notifications

This is a list of the most important built-in notifications that Zotonic sends. *Observe* (page 47) these notifications in your code to add custom functionality. Most notifications are defined as records in `include/zotonic_notifications.hrl`.

acl_is_allowed

Check if a user is authorized to perform an operation on a an object. Observe this notification to do complex or more fine-grained authorization checks than you can do through the ACL rules admin interface.

Arguments

#acl_is_allowed

action `view`, `update`, `delete` or `use`

object resource id or a module name (in case of `use`)

`Context`

Return value

false disallow operation: the default

true allow operation

undefined refrain from voting and let the next observer decide

Example

Deny anyone from viewing unpublished resource except those who have update rights on the resource (usually the creator and the administrator):

```
observe_acl_is_allowed(#acl_is_allowed{action = view, object = Id}, Context) ->
    case m_rsc:p_no_acl(Id, is_published_date, Context) of
        undefined ->
            %% Let next observer decide
            undefined;
        true ->
            %% Resource is published: let next observer decide
            undefined;
        false ->
            %% Resource is unpublished
            case z_acl:is_allowed(update, Id, Context) of
                true ->
                    %% User has update rights, so let next observer decide
                    undefined;
                false ->
                    %% Deny viewing rights on unpublished resource
                    false
            end
    end;
observe_acl_is_allowed(#acl_is_allowed{}, _Context) ->
    %% Fall through
    undefined.
```

In this observer, we return `undefined` in those cases where we do not want to deny access. We don't grant the access right away but give the next observer the change to decide whether viewing is allowed (for instance, based on the resource's category and content group and the user's group).

rsc_update

An updated resource is about to be persisted. Observe this notification to change the resource properties before they are persisted.

Arguments

`#rsc_update`

`action` Either `insert` or `update`.

`id` Id of the resource.

`props` List of resource properties.

`{IsChanged, UpdateProps}` And/remove resource properties before the update is persisted. Set `IsChanged` to `true` if you want to modify `UpdateProps`.

`Context` Site context

Example

Add a property before the resource is persisted:

```
observe_rsc_update(#rsc_update{action = insert, id = Id}, {Modified, Props},␣
↪Context) ->
    %% Set an extra property
    {true, Props ++ [{extra_property, <<"special value!">>}]}.
```

rsc_update_done

An updated resource has just been persisted. Observe this notification to execute follow-up actions for a resource update.

Arguments

`#rsc_update`

`action` Either `insert`, `update` or `delete`.

`id` Id of the resource.

`pre_is_a` List of resource categories before the update.

`post_is_a` List of resource categories after the update.

`pre_props` List of properties before the update.

`post_props` List of properties after the update.

Context Site context

Return value

`ok`

Example

Add some default edges when a resource is created:

```
observe_rsc_update_done(#rsc_update_done{action = insert, id = Id, post_is_a =␣
↪PostIsA, post_props = Props}, Context) ->
    case lists:member(activity, PostIsA) of
        false ->
            ok;
        true ->
            m_my_rsc:create_default_edges(Id, Context),
            ok
    end;
observe_rsc_update_done(#rsc_update_done{}, _Context) ->
    %% Fall through
    ok.
```

Installation requirements

Zotonic runs on Linux, Mac OS X and (not officially) on Windows.

Before running Zotonic, you must make sure your system meets the minimum requirements to do so. Zotonic needs the following software installed:

1. Erlang/OTP version **18** or newer. Build it from source, or use packages.
2. **ImageMagick** (version 6.5 or higher) for the `convert` and `identify` tools. Make sure that the convert and identify tools are in your path so that zotonic can find them. For auto-rotation to work you'll need the `exif` utility as well.
3. **PostgreSQL** version 8.4 or higher. Enable trust-authentication (username+password) in Postgres (see below).
4. **make** A recent version of the GNU `make` utility.
5. **git** Zotonic comes with a few subprojects which are pulled from the web with the `git` command.

If you meet these requirements, head straight on to *Installation* (page 7), otherwise, read on below for the specifics on these.

Erlang

You can check if you have Erlang installed by typing the following command in a terminal:

```
$ erl
```

The output should be something like:

```
Erlang/OTP 18 [erts-7.1] [source] [64-bit] [smp:4:4] [async-threads:10] [hipe]␣
↪[kernel-poll:false]
Eshell V7.1  (abort with ^G)
1>
```

(Press ctrl+c twice to exit)

If your version is below release **18**, you need to upgrade. If you don't have Erlang installed, we recommend downloading a build for your operating system from the Erlang Solutions website:

https://www.erlang-solutions.com/downloads/download-erlang-otp

ImageMagick

You can check if you have ImageMagick installed by typing the following command in a terminal:

```
$ convert -version
```

This tells you which version of ImageMagick is installed:

```
Version: ImageMagick 6.7.7-10 2012-08-17 Q16 http://www.imagemagick.org
```

PostgreSQL

You can check if you have PostgreSQL installed by typing the following command in a terminal:

```
$ psql -V
```

Returns the PostgreSQL version:

```
psql (PostgreSQL) 9.1.6
```

Enabling trust-authentication in PostgreSQL

To let Postgres users access the database from Zotonic, you need to make a configuration change in Postgres' configuration file `pg_hba.conf`.

On Linux, this file is located in `/etc/postgresql/<pg version>/main/pg_hba.conf`. Add the following lines:

```
# Zotonic settings
local   all         zotonic                           ident
host    all         zotonic     127.0.0.1/32          md5
host    all         zotonic     ::1/128               md5
```

These settings assume that your Zotonic sites are going to connect with a Postgres user called `zotonic`. For other user names, adjust accordingly. Do not forget to restart Postgres after you've made this change.

Platform-specific notes

Ubuntu / Debian

We recommend you install Erlang from the Erlang solutions website:

https://www.erlang-solutions.com/downloads/download-erlang-otp

The other requirements are easily fetched with `apt`:

```
sudo apt-get install build-essential postgresql imagemagick git
```

FreeBSD

If you're running on FreeBSD, make sure you've got the 'GNU' 'make' (check with `make --version`, which should give you GNU, and version info). If you don't have GNU make, Zotonic will give an error when trying to compile.

Mac OS X

With Homebrew you can install Erlang and ImageMagick using the following commands:

```
brew install erlang
brew install imagemagick
```

Alternatively, with MacPorts:

```
sudo port install erlang +ssl
sudo port install ImageMagick
```

For PostgreSQL choose either:

- EnterpriseDB
- Postgress.app

Windows

Currently, Zotonic is not officially supported on the Windows platform. However, the main dependencies Erlang, PostgreSQL and ImageMagick do work on Windows, so, if you're adventurous, it should be possible to get it running.

We have included user-contributed `start.cmd` and `build.cmd` batch-scripts which used to work on Windows, but have not been kept up-to-date with recent changes. Expect some major tweaking to get this ack on track.

Global configuration

This section describes the location and contents of Zotonic's *global* configuration files.

Config file locations

Zotonic depends on two global config files, called `zotonic.config` and `erlang.config`. On startup, Zotonic looks in the following places for these files:

- `$HOME/.zotonic/(nodename)/`
- `$HOME/.zotonic/(version)/`
- `$HOME/.zotonic/`
- `/etc/zotonic/(nodename)/`
- `/etc/zotonic/(version)/`
- `/etc/zotonic/`

Where *(nodename)* is the name of the Zotonic Erlang node, which defaults to *zotonic001* (and can be set with *$NODENAME* environment variable). Using the node name in the configuration path comes in handy when you want to run multiple Zotonic instances simultaneously.

(version) is the *minor* version number of Zotonic, e.g. `0.11` for all Zotonic `0.11.x` variants. This way, you can have separate configuration files for different versions of Zotonic which are running simultaneously.

When the Zotonic startup script finds a config file in one of the directories, it stops looking, so files in the other directories are ignored.

In the course of Zotonic starting up, it will print the locations of the global config files that it is using:

```
17:03:54.766 [info] Zotonic started
17:03:54.766 [info] ===============
17:03:54.766 [info] Config files used:
17:03:54.768 [info] - /home/user/.zotonic/0.11/erlang.config
17:03:54.768 [info] - /home/user/.zotonic/zotonic001/zotonic.config
```

The *zotonic.config* file

When installed for the first time, the `zotonic.config` file is well annoted with comments about what each setting does. When in doubt, consult the stock `zotonic.config` file for explanation about all config settings.

user_sites_dir, *user_modules_dir* and *user_ebin_dir*

Since version 0.11, Zotonic keeps sites and modules that are *external* to Zotonic, e.g. installed by website developers, outside the Zotonic source tree.

The directory under which Zotonic expects to find all sites is called the *User sites directory*. This is configured with the config parameter `user_sites_dir`. This directory defaults to `user/sites`, relative to Zotonic's installation directory.

The directory under which Zotonic expects to find all external modules, e.g. those installed with `zotonic modules install mod_....`, is called the User modules directory. This is configured with the config parameter `user_modules_dir`. This directory defaults to `user/modules`, relative to Zotonic's installation directory.

Optionally it is possible to change the location of the compiled erlang code in the user sites and module directories. The default location for these files is `ebin` in Zotonic's installation directory. This is configured with the `user_ebin_dir` config parameter. This makes it possible to separate the beam files of user defined code from Zotonic's beam files.

deps

The `zotonic.config` file can hold extra dependencies which are to be installed as part of a user's installation. These deps are formatted similar to how they would be listed in a `rebar.config` file:

```
{deps,
 [
  {jsx, "1.4", {git, "git://github.com/talentdeficit/jsx", {tag, "v1.4"}}}
 ]},
```

On compile time, these deps are added to the list of the standard deps from Zotonic's `rebar.config` file, and cloned and compiled in the same way.

The *erlang.config* file

The `erlang.config` file contains application environment variables for the Erlang applications that Zotonic depends on. Here you can configure for instance the paths for the log files (in the `lager` section), emqtt ports, et cetera.

Site configuration

The `config` file is the central configuration file for your site. Its syntax is the Erlang term format, essentially it is a big *proplist* with different configuration options.

New in version 0.10: Each option must be specified at most once (per file). Using a option key multiple times are no longer supported (was only used by the *hostalias* option) and any duplicates will be silently dropped. Additional config options are read from files under `config.d/`.

All files under the `config.d/` folder in the site's folder are loaded to extend/override config options from the site `config` file. So if the same key is present in both *config* and `config.d/some-file`, then the value from the latter will be used.

The files under `config.d/` are read in alphabetical order.

The following options can be configured:

`{hostname, "127.0.0.1:8000"}` This config key specifies the hostname+port part of the site's URL, to determine to which site an incoming request belongs to (since they all come in on the same port).

If you run Zotonic on port 80, or if you put a web-frontend like varnish in front of your zotonic, you can leave out the port number, and just put the hostname in there. See the *Deployment* (page 63) chapter on how to set up Zotonic for production environments.

Note: The hostname does *not* specify on which port Zotonic will listen! That information comes from the `ZOTONIC_PORT` environment variable, which, when absent, default to port 8000. Zotonic can (currently) listen on one TCP port for HTTP connections. For HTTPS, see the *mod_ssl* (page 313) chapter.

`{protocol, "http"}` This is useful for when the Zotonic is running behind a proxy (like Varnish or haproxy) and the proxy translates between HTTPS (as seen by the browser) and HTTP (as seen by Zotonic). Setting this config key to "https" ensures that redirect locations have the correct HTTPS protocol.

New in version 0.10: The `hostalias` option now holds a list of aliases.

`{hostalias, ["www.example.com"]}` The host aliases allow you to specify extra aliases for your site. This comes in handy if you have registered yoursite.com, yoursite.net and yoursite.org, and all want them to be served the same site. Mind you that Zotonic will always redirect from a hostalias to the real hostname of the site. This is done to prevent content duplication: it is good web practice to let your content live on a single URL only.

You can specify multiple host aliases; for that, just add the different *hostalias* options to the list:

```
{hostalias, ["example.com", "www.example.com",
             "example.net", "www.example.net"]},
```

`{admin_password, "test123"}` This setting specifies the password for the `admin` user. Unlike passwords for other users, the admin password is not stored in the database, but is set in the site's config file.

`{depcache_memory_max, 100}` The maximum amount of memory a site may take. The *depcache* caches various results of function calls and database queries in memory. This setting determines the maximum size of it, in megabytes.

`{redirect, true}` Whether or not to redirect the host-aliases (listed by the `hostalias` directives) to the main hostname. This defaults to true.

`{skeleton, blog}` Set by the `zotonic addsite` command, this settings tells Zotonic which skeleton site to use.

`{install_menu, [<menu-item>...]}` Creates the initial main menu when installing mod_menu. A *menu-item* is a erlang tuple with a resource id and a list of child menu-items, if any: `{rsc_name, []}`. This overrides the default menu provided by the skeleton.

`{install_modules, [<modules>...]}` List all modules that should be enabled when installing the site data. This overrides the default list of modules installed by the skeleton.

New in version 0.16: To inherit the list of modules from a skeleton, add a `{skeleton, <name>}` and it will install the list of modules from that skeleton as well.

The list of installed modules will be updated on each site start, e.g. when you add a module to the `install_modules` list, it will be installed automatically when you restart the site.

{ip_whitelist, "127.0.0.0/8,10.0.0.0/8,192.168.0.0/16,172.16.0.0/12,::1,fd00::/8"} List of TCP/IP addresses and their netmasks. The admin user password *admin* will only be accepted if logging in from a host matching the whitelisted IP addresses. This for protecting development systems that are exposed to the Internet. This can also be configured in the *Global configuration* (page 490).

{smtphost, "..."} Hostname you want e-mail messages to appear from. See *E-mail handling* (page 56).

{websockethost, "..."} The hostname that will be used for websocket requests. This hostname will be used in the browser for setting up the websocket connection. It can be used to configure a different port number for the websocket connection. For example:

```
{websockethost, "example.com:443"}
```

{cookie_domain, "..."} The domain the Zotonic session-id and page-id cookies will be set on. Defaults to the main hostname.

New in version 0.10.

{installer, <module>} Override the default zotonic installer (`z_installer`). `<module>` should make sure that the database, if used, is setup properly along with any required data. Note that it is `z_installer` that is processing the `install_modules` and `install_menu` options, so if this module is not used then those menus and modules will not be installed unless the new module performs those operations.

{service_api_cors, false} See *Uploading files* (page 53).

Database connection options

The following options for your site config specify how it connects to the database:

- dbhost
- dbport
- dbuser
- dbpassword
- dbdatabase
- dbschema
- dbdriver

These properties mostly speak for themselves, hopefully.

The *dbschema* is the name of the database schema (which is kind of a namespace for tables in Postgres); see *Tip: multiple sites using one database* below for an explanation. By default, `public` is used as the schema name.

The *dbdriver* is the name of the database driver module. Currently this defaults to `z_db_pgsql`. Other driver options are not yet implemented.

Setting module-specific config values in the site config

It is also possible to add *m_config* (page 415) values for modules to the site's `user/sitename/config` file. To do this, add clauses like this to the site's config:

```
{mod_foo, [{key, value}, ...]}
```

For instance, to set the `mod_ssl.is_secure` configuration options from *mod_ssl* (page 313), do:

```
{mod_ssl, [{is_secure, true}]}
```

Reloading the site config

After you make changes to the site config you have to restart your site for them to have effect. From the Zotonic shell, do:

```
z_sites_manager:restart(yoursitename).
```

to restart your site.

Using environment variables in the site config

Any variable in your site's `config` file can be retrieved from the OS environment variables. To do so, wrap the config value in a `{env, ...}` tuple. For instance, to use the `DB_HOST` environment variable as the database host, put the following as the `dbhost` config value:

```
{dbhost, {env, "DB_HOST"}},
```

Besides `{env, "NAME"}` tuple, you can also specify `{env, "NAME", "default value"}` for the case the environment variable is not set:

```
{dbhost, {env, "DB_HOST", "localhost"}},
```

To convert environment variables to integer (e.g. for the database port), use `env_int`:

```
{dbhost, {env_int, "DB_PORT"}},
```

or, with default value:

```
{dbhost, {env_int, "DB_PORT", "5432"}},
```

(note that the default value needs to be a string in this case, not an int).

Tip: multiple sites using one database

In Zotonic, a single PostgreSQL database can host the data of multiple web sites. This does not work using table prefixing (like Wordpress does for example), but instead, Zotonic uses Postgres' native feature *database schemas* to support this.

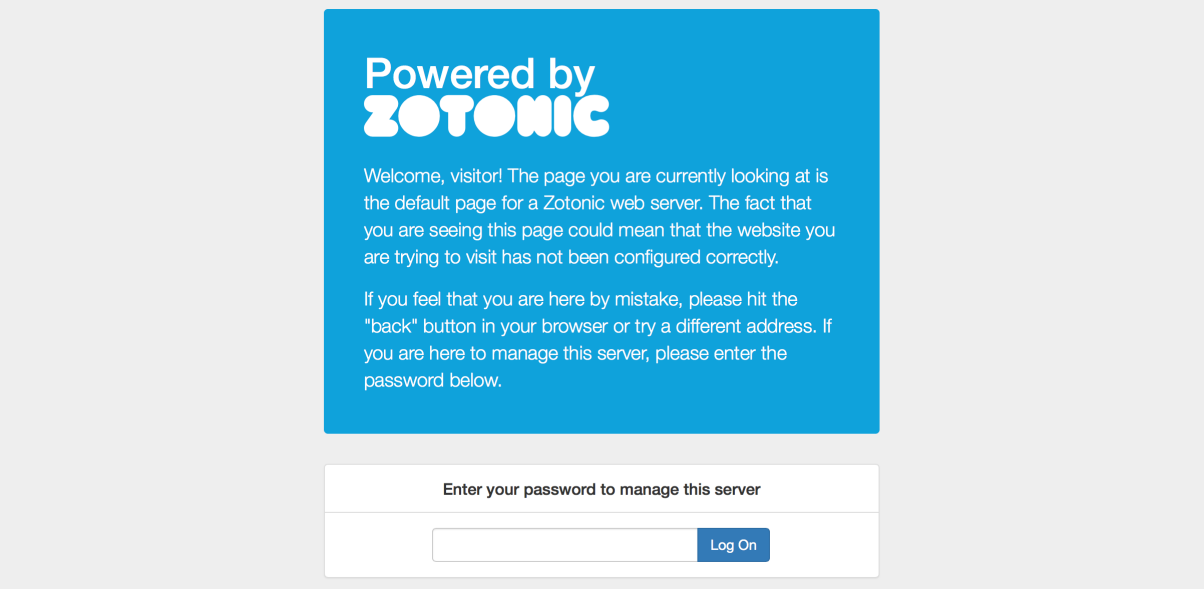
A database schema is basically another database inside your database: it's a namespace in which tables live. By default, your tables live in the namespace called *PUBLIC*, but it's quite easy to create another schema:

```
CREATE SCHEMA anothersite;
GRANT ALL ON SCHEMA anothersite TO yourdatabaseuser;
```

And then in your site config put a `{dbschema, "anothersite"}` entry next to the regular database config keys. Restart zotonic and off you go.

The Status site

The Zotonic "status" site is the first thing you see once you have installed Zotonic, when you do not have any sites configured yet. This is what it looks like:



This site is also the *fallback* site for Zotonic.

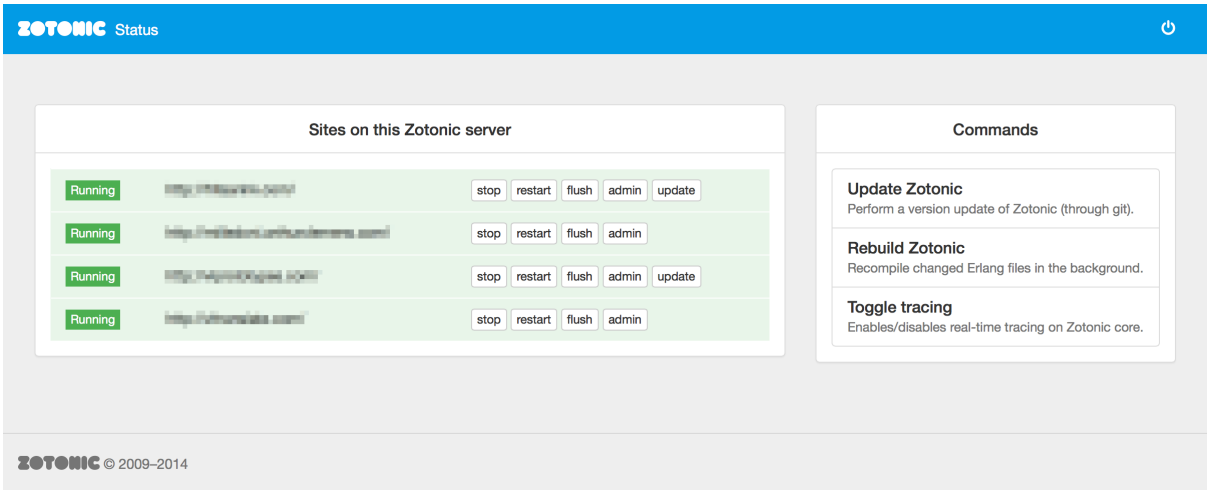
Since Zotonic supports virtual hosting, it uses the HTTP `Host:` parameter to see which site should be served at which URL. If it does not find a `Host:` header, or if the host header does not correspond to any known Zotonic site, it shows the `zotonic_status` site instead.

Logging in

When logged in to the Zotonic status site, you can manage the running sites on the system: starting, stopping and upgrading them.

Upon first visit, the site shows a friendly message, which tells visitors that the site they are looking at has probably not been configured correctly yes. It also asks for a password to log in.

The password for this site is automatically generated and stored in `~/.zotonic/[release]/zotonic.config`, for instance `[release]` is the Zotonic release number, like `0.13`.



The "update" buttons only appear when the site (or Zotonic itself) is under Mercurial or Git revision control. These buttons do a "pull" from the repository and then rebuild the system.

Getting the global sites status

The Zotonic status sites exposes a small API service which allows you to check whether all of your sites are still running:

```
http://yourzotonichost.com/api/zotonic_status/check
```

It returns a JSON response of `{"status":"ok"}` when every Zotonic site is running.

`"Running"` means that a site's status is not `"retrying"` or `"failed"`; so it does not count sites that you have manually stopped from the interface.

This API service can be plugged in to a service like https://www.pingdom.com/ to monitor the availability of all hosted sites at once.

Command-line

The `zotonic` command runs a number of utility commands which all operate on a Zotonic instance.

The `zotonic` command lives in the bin/ folder of the Zotonic source. Putting this path into your PATH variable makes working with Zotonic a lot easier:

```
export PATH=$HOME/zotonic/bin:$PATH
```

The command determines where the Zotonic base dir is by looking at its path; it always assumes that its zotonic basedir is one dir up from where the binary itself is.

Currently, the following subcommands are implemented:

`zotonic start` Start the background Zotonic server instance.

`zotonic stop` Stop the background Zotonic server instance.

`zotonic debug` Launch the Zotonic server interactively and get an EShell on the running instance. See *Command-line shell* (page 59). The `start.sh` command in the root folder is a shortcut for this command.

`zotonic restart` Restart the background Zotonic server instance.

`zotonic wait [timeout]` Wait `timeout` seconds (default: 30) for Zotonic to be started, then return.

`zotonic update` Update the server. Compiles and loads any new code, flushes caches and rescans all modules.

`zotonic shell` Get an EShell on the running instance. See *Command-line shell* (page 59).

`zotonic rpc` Send an RPC request to the running Zotonic instance. Example: *zotonic rpc "zotonic ping"*

`zotonic addsite [options] <site_name>` Creates a new site with [site_name] as its name. See guide-cli-addsite for a full overview of this command.

`zotonic startsite <site_name>` Start the site with name [site_name].

`zotonic stopsite <site_name>` Stop the site with name [site_name].

`zotonic restartsite <site_name>` Restart the site with name [site_name].

`zotonic modules <subcommand> [options]` Manages modules. It has the following subcommands:

`install <module> [module2, ...]` Installs a module from the http://modules.zotonic.com repository into your Zotonic instance. The module will be checked out using source control (either git or hg) into the priv/modules folder.

`uninstall <module> [module2, ...]` Uninstall a module

`activate <module> [module2, ...]` Activate a module

`deactivate <module> [module2, ...]` Deactivate a module

`update <module> [module2, ...]` Update a module

`restart <module> [module2, ...]` Restart a module

`reinstall <module> [module2, ...]` Reinstall a module

`sync-branches` Tries to update all installed modules to reflect the module's branch to the currently checked out Zotonic branch.

`list` List all modules available on the Zotonic Module Repository

`search <query>` Search for a module

subcommand options:

--version show program's version number and exit

-h, --help show this help message and exit

-z ZMR, --zmr=ZMR Zotonic modules repository

-s SITE, --site=SITE affected Zotonic site

-d, --debug enable debugging

-n NODE, --node=NODE Zotonic Erlang node

zotonic copysite [site_name] [source_server] Copy [site_name] and its database content from the [source_server] over SSH and load its content into the filesystem and database of the local machine. You will need to have created the database zotonic_[site_name] for this to work.

Warning: This command will reset the content of the database to the content retrieved from the [source_server]. It does, however, generate and output a restore file in case this was run by accident and explains how to recover.

zotonic createdb [site_name] Create a database called zotonic_[site_name] with the basic setup in place to host a Zotonic datastore. This script will likely need to be run as postgres unless zotonic has been granted CREATEDB in postgres as follows:

```
ALTER ROLE zotonic WITH CREATEDB
```

zotonic sitedir [site_name] Get the absolute path for a site based on [site_name]

zotonic snapshot [site_name] Take a version control snapshot of [site_name] including its database content.

This works differently from mod_backup in that it consistently uses the same filename for the SQL backup to make revision-based full site rollbacks possible.

zotonic update Update the server. Compiles and loads any new code, flushes caches and rescans all modules.

zotonic compile Compiles all the Zotonic Erlang source files, modules and sites, including those in the user sites directory and user modules directory (see *Global configuration* (page 490)). This command is mainly called from the Makefile when building Zotonic. It does *not* compile Zotonic's dependencies (the Erlang files under the `deps/` folder). This command can only be run when Zotonic is not running; for hot code reloads, use `zotonic update`.

zotonic compilefile [files...] Compiles and reloads a single *Erlang module* within the Zotonic folder. This runs very fast and works very well on a save-hook of your text editor. In Emacs, it would be called like this:

```
(add-hook 'erlang-mode-hook
      '(lambda ()
         (add-hook 'after-save-hook '
                   (lambda ()
                     (call-process "/path/to/your/bin/zotonic" nil "*scratch*" nil
↪"compilefile" buffer-file-name)
                     )
                   )
         ))
```

zotonic logtail Starts a `tail -F` on the three Zotonic log files, console.log, error.log and crash.log

Transport

The message is defined as:

```
-record(z_msg_v1, {
    qos = 0 :: 0 | 1 | 2,
    dup = false :: boolean(),
    msg_id :: binary(),
    timestamp :: pos_integer(),
    content_type = ubf :: text | javascript | json | form | ubf | atom() |␣
↪binary(),
    delegate = postback :: postback | mqtt | atom() | binary(),
    push_queue = page :: page | session | user,

    % Set by transports from user-agent to server
    ua_class=undefined :: ua_classifier:device_type() | undefined,
    session_id :: binary(),
    page_id :: binary(),

    % Payload data
    data :: any()
}).
```

The ack message is defined as:

```
-record(z_msg_ack, {
    qos = 1 :: 1 | 2,
    msg_id :: binary(),
    push_queue = page :: page | session | user,
    session_id :: binary(),
    page_id :: binary(),
    result :: any()
}).
```

Troubleshooting

Installation

Site doesn't start

Check your database connection configuration in the `config` file which is located in the *user sites directory*. By default this is `zotonic/user/sites/yoursite/config`.

Browsers can't connect to http://yoursite:8000

Check the `/etc/hosts` file and make sure you have an entry like the following:

```
127.0.0.1   yoursite
```

Zotonic won't start and shows errors when running zotonic debug

Check your site's database configuration.

Check PostgreSQL Authentication Config (`pg_hba.conf`).

If you get connection failures when starting Zotonic you should double-check pg_hba.conf and make sure to `/etc/init.d/postgresql reload` to make sure it gets loaded.

Sites

Unable to read boot script

In some cases the Zotonic nodename does not resolve well and you see an error like:

```
[error] Unable to read boot script (start_sasl.script): {error,enoent}
```

or:

```
erl_call: can't ei_gethostbyname(MacBook-Pro-Apple)
Zotonic is not running. You need to start Zotonic first to use this command.
```

Solution: Add the computer name to `/etc/hosts`, for instance:

```
127.0.0.1 MacBook-Pro-Apple
```

Erlang crashes

Here we list some common causes of Erlang crashes.

`{{badmatch,{error,emfile}}` or `{reason, emfile}`

You need to raise the *File descriptors* (page 64) limit.

EDoc reference

There are reference docs built from the source code using edoc, and may be browsed online at: http://zotonic.com/edoc/core for the core docs, and http://zotonic.com/edoc/modules for the module docs.

CHAPTER 6

Indices and tables

- genindex
- *Glossary* (page 501)
- search

Glossary

Action An action is functionality that can be attached to a HTML element or event. Actions are wired to an element or event. Think of showing dialogs, posting forms, hiding elements etc.

See also *Actions* (page 25) in the Developer Guide.

Category The data model has a hierarchical tree for the categorization of resources. Every resource is part of one category. The categorization is used amongst others to decide which template to show when displaying a resource. A category is a *resource* of the category *category*. For more information, see *Categories* (page 5).

Comet Comet is a web application model in which a long-held HTTP request allows a web server to push data to a browser, without the browser explicitly requesting it (source: Wikipedia).

Context The context is the current request context. It contains all the request data, the current site, the handle to the database and the results (scripts or templates) you will be sending back. The context is commonly passed along in Zotonic as the last argument of a function.

Controller A controller is the main entry point where a request is handled. Controllers are referenced from a dispatch rule. Commonly used controller is controller_template, which serves a template on the URL for which the controller configured. See *Controllers* (page 11).

Data model *Zotonic's generic data model* (page 3) of (categorized) resources which connect to other resources using labelled edges. This data model is loosely based on the principles of the semantic web.

Delegate A reference to a module which will be used to call a callback function on. Used in the templates when attaching actions like a :term:postback to a DOM Event. See *Actions* (page 25).

Dispatch rule A dispatch rule maps URL patterns to controllers. Dispatch rules are defined in files in the .dispatch. folder of a Zotonic module. The dispatch rule definitions are also used to generate the urls for resources and other pages. See *Dispatch rules* (page 12).

Domain model A particular configuration of resource categories and predicates, which dictate how resources of certain categories relate to each other. For example, a blog-type site might need *person*, *article* and

keyword categories, where persons and articles are connected using the *author* predicate to indicate article authorship, and articles might be connected to keywords with *has_keyword* predicates. See *The Zotonic data model* (page 3).

Edge A *resource* can connecto to other resources. These connections are called edges. Edges contain no information other than where they are linked to and from, and what their predicate is. Edges have a single direction, from the subject to the object.

Erlang module Not to be confused with a Zotonic module, an Erlang module is a single .erl file which contains Erlang functions.

Filter A template mechanism which is used inside a template to transform data before it is output. For instance: the .lower. filter transforms its input to lowercase. Filters are implemented as Erlang modules, exporting a single filter function.

Media Media are files, embed codes etc. They are attached to a resource. Every resource can hold a single medium. The resource is usually within the category *media*. See: *Media* (page 23).

Model An Erlang module which is the main accessor for retrieving data. The Erlang modules are prefixed with *m_*; in the templates they are accessible using .m... For instance, the model to access *resources* is called `m_rsc.erl`; in the template this model lets you access resources by id as `{{ m.rsc[id] }}`.

Non Informational URI The non informational uri is the base url of a resource. It always redirects to a representation of the resource. Think of a HTML page, image or JSON download. The chosen representation depends on the .Accept. HTTP request header. The non informational uri of a resource is always like http://example.com/id/1234

Page Another word for .resource.; used in the admin.

Page connection Another word for .edge.; used in the admin.

Postback An AJAX or Websocket request from the browser to the server. It is handled on the server by event/2 Erlang functions. A postback is normally sent to the controller that generated the page, but can be changed by specifying a delegate, which must be the name of an Erlang module.

Predicate Each edge has a *label* attached to it to determine what the meaning of the edge is. For instance, when an article is linked to a person, the predicate (label) might read *author*, to indicate that that person is the author of the article. A predicate is a *resource* of the category *predicate*.

Property A field in a resource. Examples are title and summary. Properties are dynamically defined. Although some property names are reserved, you can set any other property, which will be stored in the resource.

Resource The main building block of the *data model* (page 3). For simplicity of communication, a resource is often referred to as a page. Every resource usually has its own page on the web site. See *Resources* (page 16).

Scomp A scomp (from .Screen COMPonent.) is a custom template tag, implemented by an Erlang module named after the scomp name, prefixed with *scomp_*. Scomps usually generate HTML. Zotonic modules can implement their own scomp in the module.s scomps/ folder.

Service Also: API Service. Gives access to data or enables to call a function. The total set of API *calls* defines how an application responds to the outside world. Zotonic API Services provide a generalized way to create an API. API calls automatically use the authentication mechanism (session id or *OAuth* (page 306)) to perform access checks. See *API services* (page 51).

Session The session is an Erlang process. It is connected to the *session cookie* id on the browser. The session contains the id of the current user and more key/value pairs, called session variables. The session is also linked to page processes. For every open page on the browser we have a process on the server. This page process is used for the communication between the server and the user-agent (browser).

Session cookie A cookie is a small piece of data sent from a website and stored in a user's web browser while the user is browsing that website. In contrast to perisistent cookies, session cookies are created and kept only during the user's visit to the website, and deleted from the browser's cache when the user closes the session.

Tag The template systems provides tags which function as simple programming constructs. For instance, the if tag can be used for boolean tests and the for tag allows looping. The Zotonic templating system compiles

the tags found in a template to Erlang byte code which will be called when the template is rendered. This is very efficient.

Template A snippet of ErlyDTL code which is used to render a piece of content (usually HTML). Templates live under the templates/ folder of a module. The template is meant to express presentation logic.

Translation There are two kinds of translations. Texts in the templates and Erlang modules; and translations of resources. Templates and Erlang modules are translated using gettext. Resources are translated in the admin, any resource can have an arbitrary number of translations. Zotonic selects the shown language based on the preferred language of the visitor and the available languages of a resource.

User modules directory The directory in which user-installed Zotonic modules are placed. Defaults to the path `user/modules` relative to the Zotonic installation, but can be adjusted by changing the `user_modules_dir` configuration variable in the global `zotonic.config` file. See *Global configuration* (page 490).

User sites directory The directory in which user-installed Zotonic sites are placed. Defaults to the path `user/sites` relative to the Zotonic installation, but can be adjusted by changing the `user_sites_dir` configuration variable in the global `zotonic.config` file. See *Global configuration* (page 490).

Validator A validator is used to check input fields in a HTML form. A validator has two parts: the client side javascript and a server side check. You add validators to a form with the {% validate %} template tag. A validated query argument can be accessed on the server using z_context:get_q_validated/2.

Wire Connects actions and events to a HTML element. The wire scomp is the basis for most Ajax interaction on web pages. It allows to connected actions to HTML elements. Examples of actions are showing/hiding elements or postbacks to the server. ..todo:: hmm. It is scomp - it is also a often used function in the Erlang code z_render:wire/2

Zotonic module A Zotonic module (just .module., for short) is a collection of related functionality like scomps, filters, dispatch rules, controllers, templates, etc. Zotonic modules are located in folders under the modules/ directory and, by convention, are prefixed with *mod_*. See *Modules* (page 39).

Zotonic site A Zotonic site is a collection of scomps, filters, dispatch rules for one website. It is a special kind of Zotonic module with has its own config file which allows one to set the hostname, admin password, database connection parameters. It often has a set of site specific modules. The config file contains site wide settings. Zotonic uses the settings to start the site on the right port and connect it to the right database. A Zotonic system can run multiple sites.

Index

N

O

P

R

S

T

U

V

W

Z