CosCMS

diversen

Contents

A simple CMS

```
  ___|              ___|  \  |  ___|
 |      _ \   __| |     |\/ |\___ \
 |     (   |\__ \ |     |   |      |
\____|\___/ ____/\____|_|  _|_____/
```

CosCMS is a CMS with multiple export options (pdf, docx, epub, mobi, and more), and also a framework for building web applications. It is small and quite simple, so it should be easy for single persons, freelancers, and small teams to use.

Intended to use, administer, and extend with modules (web- and shell modules, and templates). For developers it is easy to export websites as simple profiles - which then can be build using the shell and git, and also put in a new repo for easy distribution.

You will need PHP >= 5.5, Apache, and a MySQL database.

Tested with Apache2 on Debian systems, and Windows. Also tested with heroku, and google app engine, PHP built-in server, and more. You should be able to make it work anywhere.

Demo

You can find a demo version at http://demo.coscms.org. Log in with

- admin / admin

Default CMS Features

This describes the **'default'** profile included with the CosCMS - which is a profile that builds a simple CMS. You will be able to remove almost any of the following modules when you have installed the system.

- The CMS in the default profile is a module, which uses other modules. The base CMS is called content.
- The CMS system uses a clone of the stackoverflow markdown editor.
- Add sub modules. Default sub modules are: comment, image, files.
- The comment module are used for adding comments to your articles. Image is for inserting inline images. Videos will be used in conjunction with the flowplayer module, if you want to enable video upload.
- All content can be filtered (articles, blog entries, comments). These filters can be removed or changed. The default filters are `markdown`, and `media` (this filter will also parse youtube, vimeo, and soundcloud URL's). These filters are included in the base package.
- The base system also includes a account module, which is used to add users, and - depending on the configuration - let users sign up.
- The account module supports email, facebook, github and openid (e.g. google) accounts (also depending on configuration). .
- The 'default' install profile also includes a nice gallery
- Included is also google analytics module, locales, menus, system (used for installing and removing modules).
- And a blog module which supports the same filters and sub modules as the content module.
- All articles and blog entries can be tagged with the tags module.
- All modules generates great SEO URLs in UTF8, and has the option for inserting a description tags.

Screenshots

Sort menu

Simple admin menu

Markdown editor

Shell mode

Sub command in shell mode

Set locales

Simple module installer

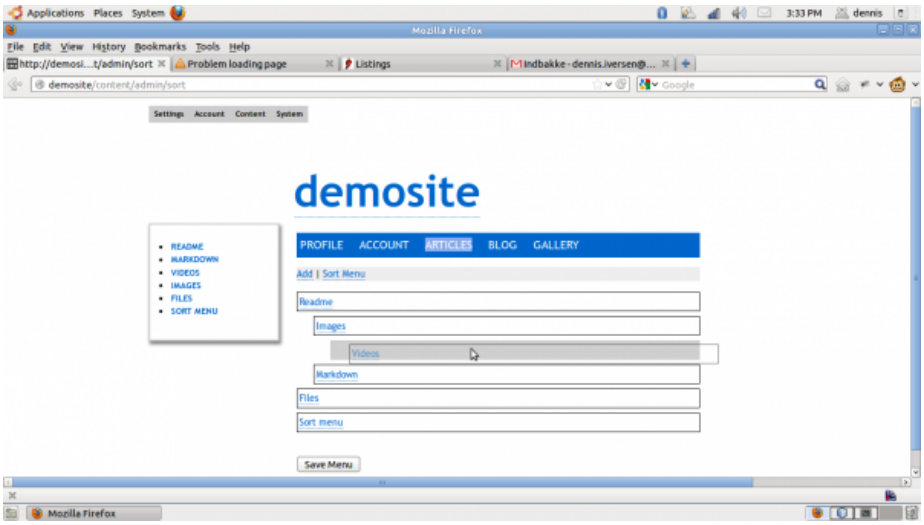

Figure 1: sort menus with jquery sortable

Figure 2: simple admin menu

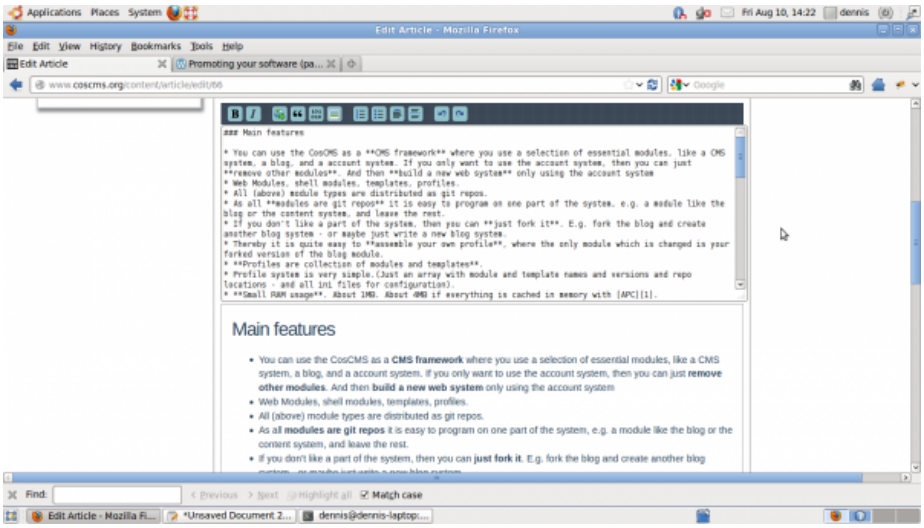

Figure 3: simple stack overflow like markdown editor

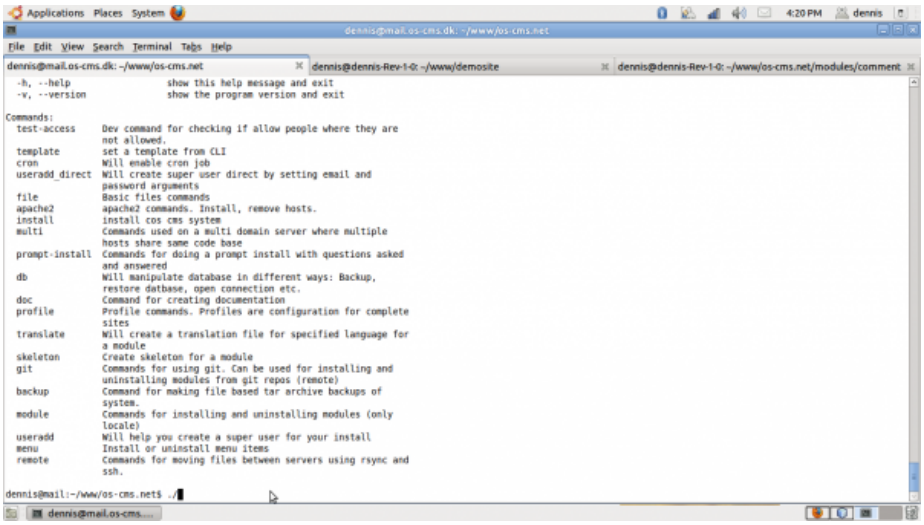

Figure 4: Shell mode

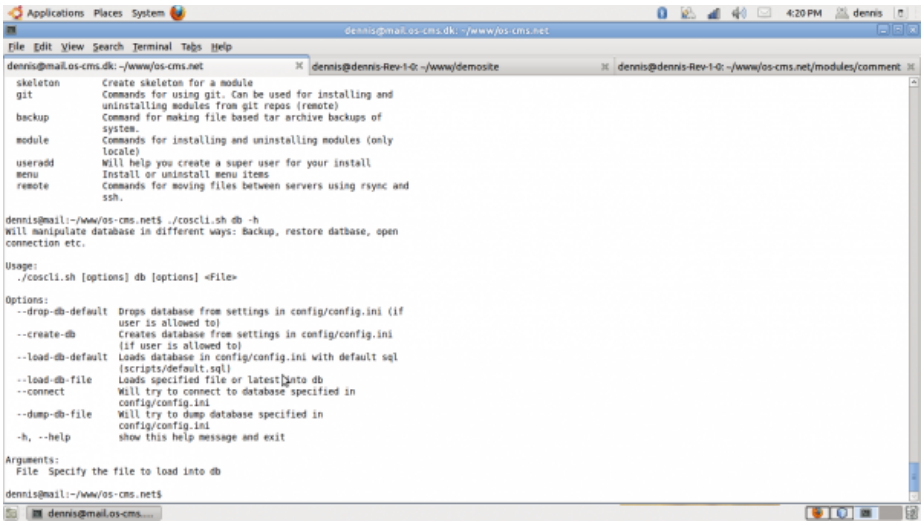

Figure 5: sub shell commands

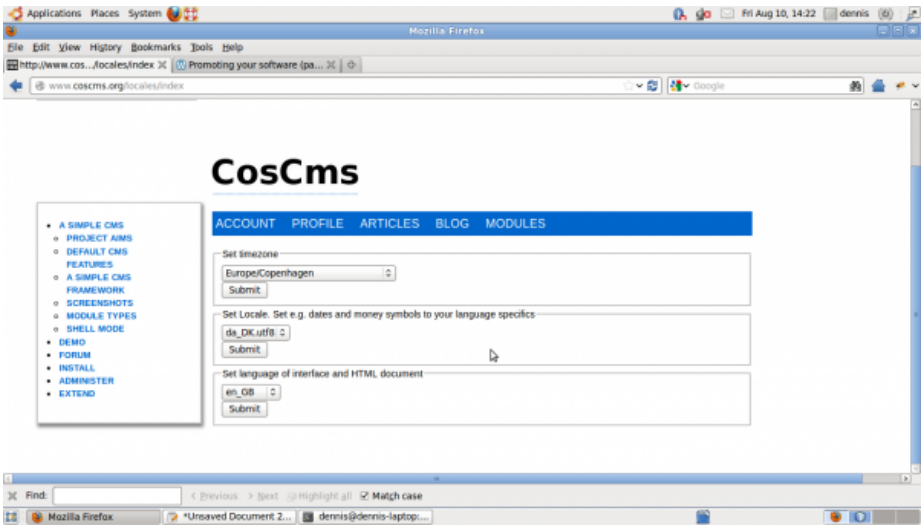

Figure 6: set locales

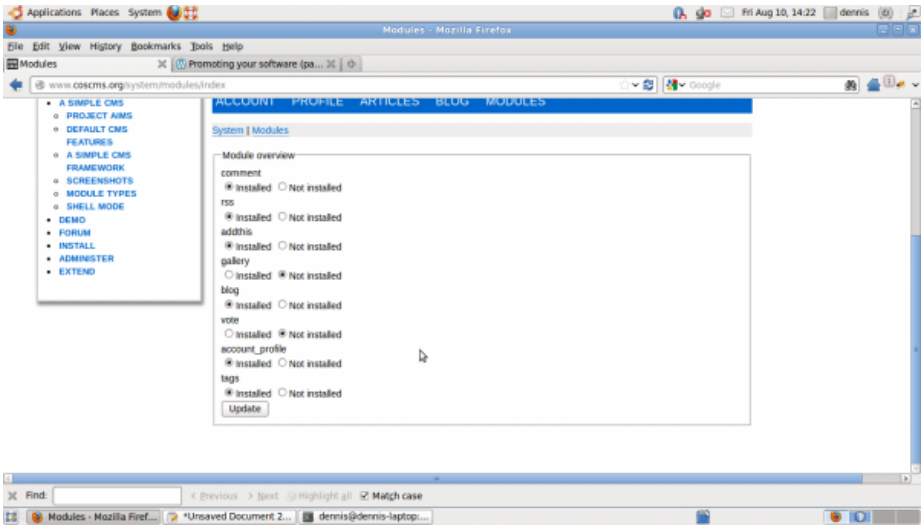

Figure 7: simple module installer

CMS Framework Features

Modules

- You can use the CosCMS as a **CMS framework** or **CMF** where you use a selection of essential modules, like a CMS system, a blog, and a account system. If you only want to use the account system, then you can just **remove other modules**. And then **build a new web system** only using the account system
- The only module you need is the system modules.

- Web Modules, shell modules, templates, profiles.
- All (above) module types are distributed as git repos.
- As all **modules are git repos** it is easy to program on one part of the system, e.g. a module like the blog or the content system, and leave the rest.
- You can also keep essential modules private while making other modules public.
- This makes it easy to engage programmers without letting them get all your source code.
- Then you easily engage a programmer for one part of the system without letting them know about other part of the system (if that's an issue).
- If you don't like a part of the system, then you can **just fork it**. E.g. fork the blog and create another blog system - or maybe just write a new blog system.

- Thereby it is quite easy to **assemble your own profile**, where the only module which is changed is your forked version of the blog module.

Profiles

- **Profiles** are collection of modules and templates.
- The profile system is very simple.(Just an array with module and template names and versions and repo locations - and all ini files for configuration).

Build system

There is also a build system where you just put all sources into one new dir - then you can make a distribution of you software quite easy.

Fast and scalable

- **Small RAM usage**. 16MB is more than sufficient for small sites.
- Use APC and everything is cached in memory - including all ini files - this will boost performance.
- Only runs modules if a page needs to load them.

Multi hosting

- Let users clone your a master site with the siteclone module. This will give them a apache2 virtual host and a database and a **complete clone of the website**.
- Let users attach own domain named to the cloned site with the domain module
- User configuration from web. Easy to attach some settings through the dbconfig module.

template system

- Simple template system, with options for adding sub templates.
- methods or views specified in a module can be overridden in a template.

Misc

- Create sub modules that are easy to use in other modules. E.g. the `comment`module which can be attached to any module you make.

- **Nice JQuery stuff**. A default markdown editor (a fork of the stackoverflow editor), some generic scripts for sorting multi level main menu, and a generic sorting script for simple lists (pull and slip).
- Base configuration from ini files.
- Menu system when writing a module - attach these to admin menu, main menu or module menus.

- **Simple separation from code and views**: place HTML in views.
- A **build system** for different type of sites.
- **Filtering and filters**
- **Translation**
- Esay to use other classes. E.g. if you don't like the default DB class you can just use e.g. redbeans or include what you need in your own module.

System dependencies

Supported Servers

You will need a web server, PHP >= 5.3, and MySQL

Working HTTP servers

It will work with Apache2 on any *nix system. It works with Apache2 on windows, but it is not well tested. It also works with the CLI build in Server for PHP >= 5.4 (this is just the servers tested). I guess it will also work on ISS, and Apache2 on AppleOS etc.

Bash LAMP install script

The following is a simple shell scripts, The first script will check and install all needed packages, which means a complete server setup including Apache2, MySQL, PHP >= 5.3, memcached, and git. And also a few PHP5 specific modules like e.g. APC (cache), GD (images) etc. For server setup on windows see this article

In order to install we need apache2, mysql-server, and php >= 5.3, git and a few more programs. You can run the following script on a recent Debian like system, e.g. Ubuntu - or just check to see what you might be missing. If you have these programs you should not need anything:

```
#!/bin/sh
aptitude install apache2-mpm-prefork
aptitude install mysql-server
aptitude install libapache2-mod-php5

# enable mod rewrite
a2enmod rewrite
```

```
# enable default image transformation tool
# GD
aptitude install php5-gd

# install php5 module for mysql
aptitude install php5-mysql

# default configuration uses memcache. If you don't install this
# you will have to change your config/config.ini file after install
# and comment out: session_handler = 'memcache'
aptitude install memcached

# install the php5 module for memcache
aptitude install php5-memcache

# install apc cache for storing all files in memory
aptitude install php-apc

# for doing development - all modules can (should) be installed with
# the {coscli.sh git} commands
aptitude install git

# We need some pear packages. Install the base package
# From version 1.9x the pear packages is included
# aptitude install php-pear
```

Note about PHP built in CLI server

On the CLI server the only thing you need is to change the files/ path owner to who runs the CLI server. Otherwise there is no issues in my 5 minutes test, quite surprised to see how well it worked, it routes all requests correct (in the same way as mod_rewrite for apache2).

If you have set your server name to e.g. coscms, then `cd coscms/htdocs` and run the following command:

```
php -S coscms:8000
```

Module Types

Web Modules

This type of modules are used for creating applications. You could build your own blog or CMS (in fact there is already a CMS system build for the system

(called 'content' - which is used on this site for writing the the content!). There is also a simple blog system. And a tutorial about building a blog.

Shell Modules

Shell modules are made in the same manner as web modules although they are used from the shell. As an example you can see the remote module, which is used to push and pull full source, web files or SQL backups with with rsync through SSH servers,

Templates

`Templates` are another kind of module: Templates are placed in webspace, which makes them great for placing html, css, images, and also PHP files (in some cases). Then you can make available some Javascript or CSS for other templates (or regular modules). And you could glue it together with a couple of PHP functions or a class.

Profiles

The third type of modules are `profiles`. Profiles are just a simple php file with a var_dump of all modules and templates, module version, and git repos, and also the .ini files for all modules. Then you can distribute your profiles (usually web applications) in a easy manner. Just clone a profile. All modules and templates will then be cloned when profile gets installed.

Simple PHP shell framework

The system is born with a set of shell commands. You can use this for e.g. installing apache2 virtual hosts. If you open a console and run the following command:

```
./coscli.sh apache2 -h
```

You will get the following options:

```
Usage:
    ./coscli.sh [options] apache2 [options] <hostname>

Options:
    --ssl       Set this flag and enable SSL mode
   --enable   Will enable current directory as an apache2 virtual host.
                Will also add new sitename to your /etc/hosts file
   --disable  Will disable current directory as an apache2 virtual host,
                and remove sitename from /etc/hosts files
    -h, --help  show this help message and exit
```

```
Arguments:
    hostname  Specify the apache hostname to be used for install or
              uninstall. yoursite will be http://yoursite
```

You can use the CosCMS if you need a simple PHP framework where you need a shell commands. Shell commands are easy to build, and you can se an example in the shell module guide

Install

There is a few ways that you can install the system. This section describes how to do it!

One package Install

Requirements

PHP >= 5.3, MySQL, Apache2

mod_rewrite have to be enabled as this will not work without. Or something similar.

Download and extract

Go to the tag list on github.

Fetch the latest.

Download zip or tar.gz file

Extract file. E.g:

```
unzip coscms-package-2.1.zip
```

copy all contents of the file into the web root

```
cp coscms-package-2.1/* htdocs/
```

Security measures.

Prevent all `.inc` and `.ini` files from being read. There is a htaccess file, which you can move to .htaccess.

This will also enable the mod_rewrite

Otherwise you will need protect these files according to your server setup.

Run Install script

Go to

`http://yoursite.org/install.php`

There is a check for the following thing:

- PDO and PDO MySQL
- Magic quotes needs to be turned off.
- files/default directory need to exist and to be writable

You will need an empty database.

Set `url`, `username`, `password` (database connection) in `config/config.ini`

Also set `server_name` and `server_redirect` (the last one you can just remove) to the name your server.

If there is a valid connection, then the system will install. Be patient as this may take a minute of so.

You will be prompted to create a user. Enter an email and a password, and you will be taken to the login screen.

Configuration

When you have logged in you can set template, mail etc. under `settings` top menu.

Windows Apache2 Easy Server Setup

Setup WAMP

This is my first install on windows 7, so it is quite brief so far.

I used this WAMP server on windows 7

My configuration files for Apache2 are attached (just above) this document.

enable mod_rewrite as it will not work without.

You will also need to enable virtual hosts, as this is the only tested working way so far.

Fetch CosCMS

Download the latest tag

Copy files

Put the coscms/* into your virtual host so that your htdocs points to your document_root

Edit config.ini

Edit config/config.ini

Set correct database user / password / database.

You can also set an SMTP email host, and many other things. But you should be OK by setting the MySQL connection string in the top.

Create database

Create a MySQL database according to your config.ini

Run install script

Go to http://dev.localhost/install.php

This will install the complete system (all tables will be created in MySQL database, so have some patience. Then the system will prompt you for creation of an email user.

Ask in the forum if any problems.

Step by Step Linux Install

Dependencies

> Note: The recommended install method is to use the `prompt-install` method as described here

Please see the dependencies before you install the system. This is the long version of how to install the coscms system on your server. There is also an install script. This is the steps that the install scripts guide you through.

Clone the Master

```
git clone git://github.com/diversen/coscms.git
```

Now you have all the source you need. If you did not specify a clone directory then the source is located in coscms. Enter this directory:

```
cd coscms
```

Now you have all the source for the base system. Checkout which tags exists by doing a:

```
git tag -l
```

Then select the latest tag or choose the master. The master is checked out when cloning, and the master is usually quite stable. If you want to set a tag then do this:

```
git checkout 1.721
```

This will produce some git output like the following: "You are in 'detached HEAD' state." And some more. Don't take notice - it is OK. It is because it is not the master. If you have chosen the master then you will not get these warnings.

Apache2 Configuration

In your /etc/apache2/sites-available/yoursite.org, You need to set the document_root in apache to coscms/htdocs e.g.:

```
/home/yourname/www/coscms/htdocs
```

You can also use this command (which will only work on Debian based systems):

```
./coscli.sh apache2 --enable yoursite.net
```

This will try to create an apache2 virtual host with correct configuration, and enable name in the /etc/hosts file. This is good for development - because then you can just use sitenames like mytestsite or costest.

Create and edit config.ini file

You now need a configuration file. In the profiles/osnet dir you will need to copy the config.ini-dist file into the config dir. Like this or in an other way:

```
cp profiles/default/config.ini-dist config/config.ini
```

Edit config/config.ini to match your database url, username and password, host (in short: all database settings). Also match enabled servername (this is for testing which user runs the server, e.g. www-data). Other settings include session time (in secounds) and cookie time (in days), default timezone. Some of these settings can be overloaded by web modules, e.g. locales.

Create database

If your db user (specified in config/config.ini) can create databases you can create the database with the following command (please note that the database and all tables will be destroyed):

```
./coscli.sh db --load-db-default
```

If your database user can not create databases, you have to create a database before install, e.g. with the MySQL command line tool, or any other way and then execute the SQL found in

```
scripts/default.sql
```

Install Profile

A profile is basically a list of modules, templates, to install, some pre-defined configuration etc. We specify the default profile included in the distribution.

```
./coscli.sh install --install default
```

Now you will see that the command line will start to clone and install all module and template repos for this profile. If you have chosen the master, then you should add the 'master' option like this:

```
./coscli.sh install --install --master default
```

Add A Super User

Execute the following command which will create a super user for you:

```
./coscli.sh useradd --super
```

Create an email user, you should be able to login at the following address:

```
http://yoursite.net/account/login/index
```

Install Non Default Profile

This install method makes it possible to install from a profile which itself is a git repo. You will first start by cloning the CosCMS by doing a:

```
git clone git://github.com/diversen/coscms.git yoursite
```

This will place the source code in yoursite dir. Next you want to cd into the profiles dir.

```
cd yoursite/profiles
```

And then you can clone your profile (this is a profile for creating static github pages from a CosCMS install):

```
git clone git@github.com:diversen/github-static-profile.git
```

Cd back to your base dir of install:

```
cd ..
```

Enable an apache2 host with the following command:

```
./coscli.sh apache2 --enable yoursite
```

Then you have a base system and a profile, an enabled website, and you can now build your site by the following command:

```
./coscli.sh prompt-install --install
```

Then your are back with the normal procedure. If your profile is a correct build you will now build a site based on the github-static-profile.

Heroku Deployment

Notices

It is easy to deploy CosCMS or a CosCMS project on heruku.

Setup

Follow the direction given here for setting up a project. I used the coscms-package, which is a package which can be pushed to heroku with ease. If you have made your own project, you can build that project, and push it to heroku.

```
git clone git://github.com/diversen/coscms-package.git
```

You will also need to make the dir files/default and chmod it to be writable, e.g:

```
mkdir files
mkdir files/default
touch files/default/test.txt
chmod 777 -R files/default
```

You will need to enable the MySQL database in heroku (there is a small free 5MB version).

Set database URL, username, password connection in `config/config.ini`. I used something like the following.

```
url = "mysql:dbname=heroku_xxxxxx;host=xxx-xxx-xxx-03.cleardb.com;charset=utf8"
db_init = "SET NAMES utf8"
username = "userb6e2999a"
password = "pass62510ccf"
```

Outcomment the default `session_handler=memcache`

```
You can keep this ini setting if you enable the memchahe extension on heroku (it is free!) -
it will speed up your application signifient. For a great boost just let this setting be,
but remember to enable the memcahe extension.
```

You will also need to move htaccess to .htaccess.

Then you just push to heroku.

```
git push heroku master
```

And go to werid-sitename.herokuapp.com/install.php

And the system should install CosCMS.

Check out my own Heroku install

mb functions and image transformation

You will need the mb functions. You can try without, but them all Redbeans dependant extensions won't work:
e.g. the hit counter.

But the mb_string extensions are easy to install. See:

https://github.com/wuputah/heroku-libraries

and for doing transformation of e.g. .jpg you will need a Imagick3 driver.

for Imagick3 driver you can find it at the following repo: https://github.com/alkhoo/heroku-cedar-php-extension

you will need to add the following to your `config/config.ini` file:

```
image_driver = 'Imagick3'
```

Add a php.ini into the `coscms-package` with the following content, for the above .so files to be loaded.:

```
extension=/app/www/mbstring.so
extension=/app/www/imagick.so
apc.rfc1867 = 1
```

The last line is for allowing displaying upload progress

Security

Depending one your version the .htaccess file may already exists, but if not, then deny access to .ini files by copying misc/htaccess to .htaccess:

```
cp misc/htaccess .htaccess
```

google app engine

You can make the CosCMS work on google app engine, but some features is missing for now. There is no image transformation libary, like GD, built-in. And I don't know how to server files from a blob - and I having looked into googles blobstore. There is also no MB libary, so modules using these features may not work. I will return to this, when I have the time.

Here is my app.yaml for know:

~.yaml application: helloworld version: 1 runtime: php api_version: 1

```
handlers:

- url: /(.*\.(htm$|html$|css$|js$|gif$|png$))
  static_files: htdocs/\1
  upload: htodcs/(.*\.(htm$|html$|css$|js$|gif$|png$))
  application_readable: true

- url: /(.*\.(ico$|jpg$|png$|gif$))
  static_files: /\1
  upload: htdocs/(.*\.(ico$|jpg$|png$|gif$))

- url: /(.*\.(png$))
  static_files: htdocs/\1
  upload: htodcs/download(.*\.(png$))
  application_readable: true

- url: /upload.php


~
```

PHP built in Server

It is possible to install CosCMS on the PHP built in server which is enabled from PHP >= 5.4.

- You will need to chmod the files/default dir to the Cli User.

Beside from this you should just follow one of the other directions.

And then run the following command in base htdocs path:

```
php -S coscms:8000
```

Linux / Ubuntu install

This says Ubuntu install (as this is the system I am currently working on), but it should work on any *nix system if you have met the dependencies.

Check the dependencies.

First clone the base system:

```
git clone git://github.com/diversen/coscms.git yoursite
```

Enter base system:

```
cd yoursite
```

Enable apache2 host

```
// you will need to be root
sudo ./coscli.sh apache2 --en yoursite
```

Ready to install. You will be asked about DB configuration. Will ask you for version to install. Check out the `latest version` version or try `master` (tagged versions are tested and should work, while master will work 99% of the time work as well). After writing the `config/config.ini` file and install all profile's modules from git repos. At last system will prompt you for a super user.

```
./coscli.sh prompt-install --install
```

Set correct perms for public files (e.g. upload folder)

```
// you will need to be root user as we change
// the perms to be www-data
sudo ./coscli.sh file --chmod-files
```

Shell Usage

Many admin options can only be controlled through CLI mode. Cd into the base dir of your installed website and run the following command.

```
./coscli.sh -h
```

For futher help use, e.g. for git:

```
./coscli.sh git -h
```

And you will get a list of options to use.

Upgrade

cd into your base dir:

```
cd www/website
```

checkout new version, e.g. 1.822:

```
./coscli.sh upgrade --up
```

It will examine if there are any new tags, and then upgrade system, all modules, and reload configuration files.

Prevent .ini override

In order to prevent changes to ini files when upgrading, you will need to add a locale.ini where you change your ini overrides.

E.g. in account. The default profile uses email only as a login:

In order to override this you can make a `locale.ini` file in your account modules containing the following lines (which also will enable facebook login).

```
account_logins[0] = "email"
account_logins[1] = "facebook"
```

In this fashion you don't need to worry about personel ini settings you have made.

Shell commands

CosCMS comes with a few built in commands, and at the same time it is easy to extend.

If you do a

```
./coscli.sh -h
```

somthing like the following will be printed:

```
                  _ _       _
  ___ ___  ___  ___| (_)  ___| |__
 / __/ _ \/ __|/ __| | | / __| '_ \
| (_| (_) \__ \ (__| | |_\__ \ | | |
 \___\___/|___/\___|_|_(_)___/_| |_|

    Modulized Command line program


Usage:
  ./coscli.sh [options]
  ./coscli.sh [options] <command> [options] [args]

Options:
 -d domain, --domain=domain  Domain to use if using multi hosts. If not
                             set we will use default domain
  -v, --verbose              Produce extra output
  -h, --help                 show this help message and exit
  --version                  show the program version and exit

Commands:
  apache2          apache2 commands. Install, remove hosts.
  backup        Command for creating and restoring file based tar.gz
                   archives of a complete system.
  build        Command for building a disto based on a single archive .
  cache            Commands for clearing caches.
  db             Will manipulate database in different ways: Backup,
                   restore datbase, open connection etc.
  dev              Dev commands for testing and checking.
  doc              Command for creating documentation
  file         Basic files commands for setting correct permissions and
                   creating essential files like logs
  git           Commands for using git. Can be used for installing and
                   uninstalling modules from git repos (remote)
  install       install CosCMS from a specified profile [prompt-install
                   is recommended for install]]
  menu             Install or uninstall menu items
  module        Commands for installing and uninstalling modules - only
                locale placed in module dir. For remote install of git
                   repos see git
  multi         Commands used on a multiple domains found in same path
  multi-shared   Commands used on a multi domain server where multiple
                   hosts share same code base
  profile       Profile commands. Profiles are configuration for complete
              sites. You can build profiles for distribution, install
                   from profiles, reload configuration and more.
```

```
 prompt-install  Commands for doing a prompt install with questions asked
                   and answered
  skeleton        Create skeleton for a module
  template        set a template from CLI
 translate     Will create a translation file for specified language for
                   a module
  upgrade         Upgrade a existing system
  useradd        Will help you create a super user for your install
  useradd-direct  Will create super user direct by setting email and
                   password arguments
 remote       Commands for moving files between servers using rsync and
                   ssh.
```

For getting help on any sub command use e.g:

```
./coscli.sh db -h
```

This will output the following:

```
Will manipulate database in different ways: Backup, restore datbase, open
connection etc.

Usage:
  ./coscli.sh [options] db [options] [File]

Options:
  --show-con-info    Show DB connection info
 --drop-db-default  Drops database from settings in config/config.ini (if
                     user is allowed to)
 --create-db     Creates database from settings in config/config.ini
                (if user is allowed to). Warning: Will drop database
                     if it exists, and create database again.
 --load-db-default  Loads database in config/config.ini with default sql
                   (scripts/default.sql). Only the very basic SQL is
                     loaded into database.
 --load-db-file    Loads specified file or latest found in backup/sql
                     into db
 --connect       Connects to database specified in config/config.ini
                     using the mysql shell
  --dump-db-file     Will try to dump database specified in
                config/config.ini. If no file is specified then dump
                     will be generated with a unix timestamp.
  -h, --help         show this help message and exit

Arguments:
  File  Optional: Specify a path to a file to load into db
```

Extend

This section holds information about extending the coscms. You can write your own templates, create a profile or create a module...

Web Module Guide

The aim of this page is to show the steps in developing a module. The module we will be making here is a simple blog system which can be used by administer users.

It is divided into tree parts and you should be able to complete the tutorial in an hour or so.

If you just want the code you can cd into the module dir and clone the source:

```
git clone https://github.com/diversen/myblog.git
```

Or install it as a module in your CosCMS by doing a:

```
./coscli.sh git --mod-in https://github.com/diversen/myblog.git
```

Or install the master:

```
./coscli.sh git --mod-in --master git://github.com/diversen/myblog.git
```

Or you can just view the source code on github, at https://github.com/diversen/myblog.git:

Prerequisite

Module Files A module is a dir with some files:

create a directory inside the modules dir called `myblog`.

```
mkdir modules/myblog
```

For our blog example we will use SQL. Let us create some SQL as the first task:

SQL Files We start creating some SQL for storing the blog posts. Create some dirs like this:

```
mkdir modules/myblog/mysql
mkdir modules/myblog/mysql/up
mkdir modules/myblog/mysql/down
```

Open a file called 1.01.sql

```
./modules/myblog/mysql/up/1.01.sql
```

In this file we will place your SQL which will be executed when we install version 6.01 of our module. The mysql/up and mysql/down are used when we want to install, uninstall, or upgrade modules. Open the above file in a text editor and place the following SQL:

```
DROP TABLE IF EXISTS `myblog`;

CREATE TABLE `myblog` (
  `id` int(10) NOT NULL AUTO_INCREMENT,
  `title` varchar(128) NOT NULL,
  `entry` text,
  `updated` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
  `user_id` int(10) unsigned DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=UTF8;
```

As you should notice there is a line between each statement. You should remember this because each statement is executed after one and another. So `always remember a newline between each statements`.

The table contains a field for `title`, a field for the blog `entry` and a `updated` timestamp field for update information, and a `user_id` field for keeping order about which user has made the blog entry. We also make a SQL file for removing the table on uninstall. Open this file:

```
./modules/myblog/mysql/down/1.01.sql
```

In this file we place the following sql:

```
DROP TABLE IF EXISTS `myblog`;
```

Install File Now we have some SQL, and we would like to install this SQL (and the module). In order for doing so, we open our install file and sets the current version for the module. Open:

```
./modules/myblog/install.inc
```

And place the following php code in it:

```
<?php

namespace modules\myblog;
use diversen\lang;

// This sets the version of the module.
// Note that the version follows sql.
//
// If we wanted to update our SQL at a later point
// we will add a fill to the 'up' dir where we place a
// e.g. 1.02.sql file and update $_INSTALL['VERSION] as well.
$_INSTALL['VERSION'] = 1.7;

// This creates a main menu item in the database
$_INSTALL['MAIN_MENU_ITEM'] = array (
    // menu items are not translated before being inserted in menus table
    // therefor the 3. param with ('no_translate' => true)
    'title' => lang::translate('My blog', null,
            array ('no_translate' => true)),
    'url' => '/myblog/index',
    'module_name' => 'myblog',
    'parent' => 0,
    'weight' => 2,
);

// This sets the public clone URL
$_INSTALL['PUBLIC_CLONE_URL'] = "git://github.com/diversen/myblog.git";
$_INSTALL['PRIVATE_CLONE_URL'] = 'git@github.com:diversen/myblog.git';
```

Install Module Now we have a module which we can install. To install a module which is already located in the module folder we call this command:

```
./coscli.sh module --mod-in myblog
```

This will install the module.

Lets just try and uninstall again right away.

```
./coscli.sh module --mod-down myblog
```

This just install, and uninstall the SQL.

Module Menu In the above the main menu item was installed (which is placed in database). You will notice that our menu is prefixed with a `NT:` (`needs translation`). This is because we have not translated our menu item (we will do that as a final thing) - if you don't need translation you can just `echo` statements without using the translation class.

The next menu structure belongs only to the myblog module which means that it will only be loaded when our module gets loaded. The module menu is also just a file. Open the file called myblog/menu.inc. Add the following PHP code to it:

```php
<?php

namespace modules\myblog;

use diversen\conf;
use diversen\lang;
/**
 * This is where we define our menus.
 */

// First menu item is where we plan to list all entries.
// As you can see the module menu just specifies links
// to controller files. index refers to index.php in our
// module folder
$_MODULE_MENU[] = array(
    'title' => lang::translate('View all'),
    'url' => '/myblog/index',
);

// we read who is allowed to use admin interface
// which we get from an ini setting. We will see this ini
// file in the next section of the tutorial.

$myblog_allow = conf::getModuleIni('myblog_allow');

// we add a menu entry for adding entries. Again it is just
// a link with URL and title, but notice that we set an auth
// element of the array in addition.
// This just means that it is only users logged in as admin
// users who will see this link. Anybody else does not need to
```

```
// see this link.
$_MODULE_MENU[] = array(
    'title' => lang::translate('Add'),
    'url' => '/myblog/add',
    // note that we have set an auth element!
    'auth' => $myblog_allow
);
```

Adding ini settings

Before we go on and produce the module, we will add a couple of ini settings - these are good if you want to be able to edit some configuration from a central point - which can be changed: Open myblog.ini, and add the following lines of code:

```
myblog_per_page = 5
myblog_filters[0] = cosmarkdown
myblog_abstract_length = 250
myblog_allow = admin
```

Adding the module

The following is the full listing of the code which make up our module

```
<?php

namespace modules\myblog;

use diversen\db;
use diversen\db\q;
use diversen\html;
use diversen\http;
use diversen\lang;
use diversen\session;
use diversen\template;
use diversen\uri;
use diversen\view;
use diversen\pagination;
use diversen\html\helpers as htmlHelpers;

use modules\myblog\views;

/**
 * First thing is a nice trick
 * .
```

```
 * when making the next call we make it possible for all 'views' to be
 * overriden in a template.
 *
 * All views in this example are just functions calls,
 * which makes it easy to navigate in e.g. and IDE (like netbeans).
 * You could also place views in a view folder and call them there -
 * but this is what I prefer.
 */
view::includeOverrideFunctions('myblog', 'views.php');

class module {

    /**
     * variable holding error codes and messages.
     * @var array   array for holding error codes
     */
    public $errors = array();


    /**
     * method for getting a filtered entry id.
     * @return int  $int returns filtered int
     */
    public function getEntryId (){

        // We will use the uri class for getting info about the
        // URI. The first fragment [0] is the module name
        // the next [1] is the action name, and the third[2] is
        // an ID identifying the id of the blog post when we will read,
        // update or delete a post.
        //
        // in this sense our url will look something like this:
        // myblog/index/123
        //
        $id = uri::fragment(2);

        // we validate it as int before returning it.
        return filter_var($id, FILTER_VALIDATE_INT);
    }

    /**
     * function for creating a form for insert, update and deleting
     * blog entries. This could have been place in the myblog_views
     * class as we then would be able to override this in a template.
     *
     * @param string    $method (update, delete or insert)
```

```
     * @param int       $id (if delete or update)
     * @param array     $values array with values if we load the form for
     *                  an update operation
     */
    public function form($method, $id = null, $values = array()){

        // we use html.php  from emthods, which is easy to use for creating
        // forms connected to a database
        //
        // You don't need to use this class when you make your
        // modules. In fact you could use the Zend Form,
        // Your own form class, or just write
        // the form as HTML and place it somewhere in the module dir.
        //
        // Though: For the uniform look and feel of a form, using these class
        // makes it quite easy to manipulate looks of the forms in the template.

        if (isset($id)){
            // if an id is set if update or delete
            if ($method == 'delete'){

                // simple helper to displat a delete form
                echo htmlHelpers::confirmDeleteForm('submit',
                        lang::translate('Delete entry'));
                return;
            } else {
                // edit form
                // select one (the same syntax as the above method, but this time
                // params ('table' 'fields to fetch' simple search e.g. array('id' => 12
                $db = new db();
                $values = $db->selectOne('myblog', 'id', self::getEntryId());
                $values = html::specialEncode($values);
                $legend = lang::translate('Edit entry');
            }
        } else {
            $legend = lang::translate('Add entry');
        }

        $form = new html();
        $form->formStart();

        // form will use submitted variables
        // if $_POST['submit'] is set

        $form->init($values, 'submit');
        $form->legend($legend);
```

```
    $form->label('title', lang::translate('Title'));
    $form->text('title');
    $form->label('entry', lang::translate('Entry'));
    $form->textarea('entry');
    $form->submit('submit', lang::translate('Submit'));
    $form->formEnd();
    echo $form->getStr();
}



/**
 * we add a method for validating the submitted entry
 */
public function validate(){
    if (isset($_POST['submit'])){
        // very simple check. We just make sure something is
        // added to title and entry fields.
        if (empty($_POST['title'])) {
            $this->errors[] = lang::translate('No title');
        }
        if (empty($_POST['entry'])) {
            $this->errors[] = lang::translate('No entry');
        }
    }
}


/**
 * we add a method for santizing the submitted entry
 * on every submission this is called
 */
public function sanitize(){
    if (isset($_POST['submit'])){
        // we rewrite htmlentites
        $_POST = html::specialEncode($_POST);

    }
}


/**
 * method for inserting a blog entry into database
 *
 * @return  boolean true on succes false on failure
 */
```

```
public function addEntry (){

    // we add user id and updated fields to the _POST var
    $_POST['user_id'] = session::getUserId();

    // all values will be prepared
    // this just removes fields like 'submit' and 'captcha' from the
    // form submission. Also note that we have performed
    // htmlentites on all fields

    $values = db::prepareToPost();

    // prepare and execute
    // insert the blog post

    $db = new db();
    $res = $db->insert('myblog', $values);

    // return boolean result from insert operation
    return $res;
}

/**
 * method for updating an entry
 * @return boolean
 */
public function updateEntry (){

    // we add user id and updated fields to the _POST var
    $_POST['user_id'] = session::getUserId();

    // all values will be prepared
    // this just removes fields like 'submit' and 'captcha' from the
    // form submission
    $values = db::prepareToPost();

    // prepare and execute
    // insert the blog post

    $db = new db();
    $res = $db->update('myblog', $values, self::getEntryId());

    // return boolean result from insert operation
    return $res;
}
```

```php
/**
 * method for updating an entry
 * @return boolean
 */
public function deleteEntry (){
    $db = new db();
    $res = $db->delete('myblog', 'id', self::getEntryId());
    return $res;
}

/**
 * method for listing latest entries in our blog
 *
 * the controller file is placed in top of our module dir, and
 * it will be called index.php which refers to this URL:
 *
 * myblog/index
 *
 */
public function indexAction (){

    // set title of html document
    template::setTitle(lang::translate('View all blog entries'));

    // We define how many items we want to show per page.
    // This is defined in our ini file.
    //
    $per_page = 10;

    // get a count of all rows in myblog_table
    $num_rows = q::numRows('myblog')->
            fetch();

    // all you need to tell pearPager object is the count of all numRows
    $pager = new pagination($num_rows, $per_page);

    // Use the simple query biulder - q
    $rows = q::select('myblog')->
            order('updated')->
            limit($pager->from, $per_page)->
            fetch();

    // encode rows
    $rows = html::specialEncode($rows);

    // display the rows
```

```
        // see: views.php

        views::index($rows);

        // and we print the pagination data.
        echo $pager->getPagerHTML();

    }

    /**
     * method for listing single full entry of our blog
     *
     * it displays a full blog entry in this style , e.g.:
     *
     * myblog/view/123
     *
     * which will display the blog entry 123 and display it
     * for the reader
     *
     */
    public function viewAction (){

        // create a db object
        $db = new db();

        // select one
        $row = $db->selectOne(
                'myblog',
                'id',
                self::getEntryId());

        // we set a html title for the page where this method is used
        template::setTitle(
                lang::translate('View entry') ) .
                MENU_SUB_SEPARATOR_SEC .
                $row['title'];

        // call the 'view' function
        views::view($row);

    }

    /**
     * add controller
     *
     * method for adding a blog entry. This is the add controller.
```

```
 * This is is used when user hits /myblog/add
 *
 * @return void
 */
public function addAction(){

    // we set a title for the page where this method is used
    // this is the <title> tag. in the page.
    template::setTitle(
            lang::translate('Add entry'));

    // we check if user trying to add a blog entry is allowed to
    // if not we just return
    //
    // the session::checkAccessControl ('level') checks myblog.ini
    // to see what the setting myblog_edit equals. In our case
    // it equals 'admin'. Instead of setting the access level directly
    // we can now just edit our access restrictions in the myblog.ini
    // file. The session::checkAccessControl will also flag a 403 if
    // the user in session does not meet the required access level.
    // and will be redirected to a 403 error file.

    if (!session::checkAccess('admin')){
        return;
    }

    // if any a form is submitted we check for errors and add the
    // the entry
    if (isset($_POST['submit'])){

        // validate and sanitize (see methods above)
        $this->sanitize();
        $this->validate();

        // if no errors
        if (empty($this->errors)){

            // add entry, but decode htmlentities.
            $_POST = html::specialDecode($_POST);
            $res = $this->addEntry();

            // if success in adding entry - redirect
            if ($res){
                http::locationHeader(
                        "/myblog/index",
                        lang::translate('Entry created')
```

```
                        );
                }
            // if errors we display the erros
            } else {
                // we use the view_form_errors, which is one of several
                // helper functions which can be added to a template file.
                html::errors($this->errors);

                // display form after errors
                $this->form('insert');
            }
        } else {
            // no submisstion show blog entry form
            // the form will set correct values if form has
            // been submitted once
            $this->form('insert');
        }
    }

    /**
     * edit controller
     * @return void
     */

    public function editAction (){
        // we set a title for the page where ths method is used
        template::setTitle(lang::translate('Update entry'));

        // we check if user is trying to add a blog entry is allowed to
        // if not we just return
        if (!session::checkAccess('admin')){
            return;
        }

        // if any a form is submitted we check for errors and add the
        // the entry
        if (isset($_POST['submit'])){
            $this->validate();
            $this->sanitize();
            if (empty($this->errors)){

                // decode htmlentities again.
                $_POST = html::specialDecode($_POST);
                $res = $this->updateEntry();
                if ($res){
                    // redirect and set action message
```

```
                // which will be displayed on next page.
                http::locationHeader(
                        '/myblog/index',
                        lang::translate('Entry updated'));
            }
        } else {
            html::errors($this->errors);

        }
    }
    $this->form('update', self::getEntryId());
}

/**
 * delete controller
 *
 * @return void
 */
public function deleteAction (){

    // we set a title for the page where ths method is used
    template::setTitle(lang::translate('Delete entry'));

    // we check if user is trying to add a blog entry is allowed to
    // if not we just return
    if (!session::checkAccess('admin')){
        return;
    }

    // if any a form is submitted we check for errors and add the
    // the entry

    if (isset($_POST['submit'])){
        $res = $this->deleteEntry();
        if ($res){
            http::locationHeader(
                    "/myblog/index",
                    lang::translate('Entry deleted')
                );
        }
    } else {
        // no submisstion show blog entry form
        // the form will set correct values if form has
        // been submitted
        $this->form('delete', self::getEntryId());
    }
```

```
    }
}
```

Views

All views are include in the file in the top of the module:

```
<?php

namespace modules\myblog;

use diversen\conf;
use diversen\html;
use diversen\lang;
use diversen\moduleloader;
use diversen\session;
use diversen\strings;
use diversen\time;
use diversen\user;

/**
 * view methods are placed in myblog_views class
 */
class views {

    /**
     * view for index action
     * @param array $vars
     */
    public static function index ($vars = array ()) {
        foreach ($vars as $row) {

            $link_url = "/myblog/view/$row[id]/";
            $link_url.= strings::utf8SlugString($row['title']);
            $link = html::createLink($link_url, $row['title']);

            // we print headline for our blog entry
            html::headline ($link);

            $date = time::getDateString($row['updated']);
            echo user::getProfile($row['user_id'], $date);

            // we fetch a module setting telling us how long our abstract
            // should be
            $teaser_length = 250;
```

```
            // we use the standard function substr2 to get our teaser of the entry
            $teaser = strings::substr2($row['entry'], $teaser_length, 3);

            // print teaser to screen
            echo "<p>$teaser</p>\n";
            self::displayEditOptions ($row);
        }
    }

    public static function displayEditOptions($row) {
        // we check if user has access right to view admin options of this
        // entry. (as a standard all access control function can be found
        // in the session class located in lib/session.php
        $ret = session::checkAccessControl('myblog_allow', false);

        // if access is ok we print some admin links.
        if ($ret) {
            echo html::createLink(
                    "/myblog/edit/$row[id]", lang::translate('Edit'));
            echo MENU_SUB_SEPARATOR;
            echo html::createLink(
                    "/myblog/delete/$row[id]", lang::translate('Delete'));
        }
    }

    /**
     * view for view action
     * @param array $vars
     */
    public static function view ($vars) {

        // echo title
        // we get filters set in myblog.ini
        // markdown (located in modules/filter_markdown/markdown.inc)
        $filters = conf::getModuleIni('myblog_filters');

        // we filter content with a helper function called get_filtered_entry
        // which filters entry with all filters specified in the array $filters
        $vars['entry'] = moduleloader::getFilteredContent(
                $filters,
                $vars['entry']);


        html::headline(html::specialEncode($vars['title']));
```

```
        // compose formatted date as string
        // $datetime = strtotime($vars['updated']);
        $date_formatted = time::getDateString($vars['updated']);

        // get simple profile
        echo user::getProfile($vars['user_id'], $date_formatted);

        // echo entry
        echo $vars['entry'];

        // display edit options
        self::displayEditOptions($vars);
    }
}
```

Now we have everything except the controllers which ties it all together in some different files.

Translate

Translation Now we have our module. We have authentication, pagination, and a simple blog for admin users. It would be nice to translate all strings specified with the `lang::translate` static call. Easy enough, if you can use this command:

```
./coscli.sh translate --translate myblog en
```

This will create a `language.php` file which is auto generated and located in `myblog/lang/en/language.php`.

```
<?php

$LANG = array();

// Translation of file modules/myblog/module.php

$LANG['Delete entry'] = 'Delete entry';
$LANG['Edit entry'] = 'Edit entry';
$LANG['Add entry'] = 'Add entry';
$LANG['Title'] = 'Title';
$LANG['Entry'] = 'Entry';
$LANG['Submit'] = 'Submit';
$LANG['No title'] = 'No title';
$LANG['No entry'] = 'No entry';
```

```
$LANG['View all blog entries'] = 'View all blog entries';
$LANG['View entry'] = 'View entry';
$LANG['Entry created'] = 'Entry created';
$LANG['Update entry'] = 'Update entry';
$LANG['Entry updated'] = 'Entry updated';
$LANG['Entry deleted'] = 'Entry deleted';
// Translation of file modules/myblog/install.inc

$LANG['My blog'] = 'My blog';
// Translation of file modules/myblog/views.php

$LANG['Edit'] = 'Edit';
$LANG['Delete'] = 'Delete';
// Translation of file modules/myblog/menu.inc

$LANG['View all'] = 'View all';
$LANG['Add'] = 'Add';
```

As you see it is just an array with keys that are the same as the array values. If you want to create eg. a danish translation, you would just create a `da` dir and copy the en/language.php file into it, and translate to danish.

Shell Module Guide

What we will make

This is very simple. We want to make a shell module that can clone a site with wget and make a static site which we then can publish on github.com. For an example you can see my personal github site which is made from the following module. Se http://diversen.github.com This is a static site made from a dynamic site.

If you just want to look at the source you can do a:

```
./coscli.sh git --mod-in git://github.com/diversen/gh-static.git
```

Install File

First step is to make a install file

```
<?php

// first we set a version for our module Note it has to be a float!
$_INSTALL['VERSION'] = 2.1;
```

```
// we specify that it is a shell module.
// this means that the source files will be loaded when the base shell
// command is getting loaded.
$_INSTALL['IS_SHELL'] = "1";

// we add a public clone url for easy distribution
$_INSTALL['PUBLIC_CLONE_URL'] = 'git://github.com/diversen/gh-static.git';

// and also a private url if we want to commit directly from base
// shell command
$_INSTALL['PRIVATE_CLONE_URL'] = 'git@github.com:diversen/gh-static.git';
```

Ini File

We also make a ini file where we can add some settings which can be changed with ease. It looks like this:

```
gh_static_push_url = "git@github.com:diversen/diversen.github.com.git"
gh_static_site_dir = "/home/dennis/www/diversen.github.com"
gh_static_index_url = "cos.org"
```

The first setting of the ini file is the push url. This is the git repo url we can push to. The next setting is the dir where our site will be placed local. Third setting is the url to index: mygithub (a more normal url to spider would be e.g. www.example.com).

The module file

Then we can make our module:

```
<?php

/**
 * This is function which genrates the static site.
 */
function gh_static_generate (){

    // ini settings will be found in the static var mainClin::$ini
    $static_dir = config::getModuleIni('gh_static_site_dir');

    // The url to index (mygithub)
    $url = config::getModuleIni('gh_static_index_url');
```

```
    // create dir where files will be put
    if (!file_exists($static_dir)) {
        $command = "mkdir $static_dir";
        cos_exec($command);
    }

    // wget command
    $command = "wget -m -k -K -E  $url";

    // execute
    cos_exec($command);

    // copy the content of the url to the static dir
    // (/home/dennis/www/diversen.github.com)
    //$command = "mkdir $static_dir";


    $command = "cp -rf ./$url/* $static_dir";

    // execute
    system($command);
}

/**
 * next command is the push command.
 */
function gh_static_push (){

    // the static dir where we will push from
    $static_dir = config::getModuleIni('gh_static_site_dir');

    // cd into dir add - commit - push
    $command = "cd $static_dir && git add . && git commit -m \"auto-commit\" && git push

    // execute
    cos_exec($command);
}


// enable commands in shell - first we set a base command
mainCli::setCommand('gh-static', array(
    'description' => 'Command to generate a static site from a dynamic',
));

// then we set a option
mainCli::setOption('gh_static_generate', array(
```

```
    'long_name'   => '--generate',
    'description' => 'Will generate the static site into location specified in .ini file
    'action'      => 'StoreTrue'
));

// and another option
mainCli::setOption('gh_static_push', array(
    'long_name'   => '--push',
    'description' => 'Will push the static site to git url specified in .ini file',
    'action'      => 'StoreTrue'
));
```

Usage

`./coscli.sh gh-static -h`

Will give you the help options.

`./coscli.sh gh-static --generate`

Will generate the static site with wget

`./coscli.sh gh-static --push`

Will push the site to remote repo.

Simple as that!