Drupal notes:

Contents

[1. Database : drupal712 4](#_Toc323382350)

[1.1. Database user and password drupaluser/1234 4](#_Toc323382351)

[1.2. url:http://localhost:81/drupal7 4](#_Toc323382352)

[1.3. Site admin : admin/680628 4](#_Toc323382353)

[2. Create a git repo: 4](#_Toc323382354)

[2.1. Right click the folder, create repository 4](#_Toc323382355)

[2.2. Add files to repo 4](#_Toc323382356)

[2.3. Go on to github ,create a repo 4](#_Toc323382357)

[2.4. On desktop, create a remote with the url like 4](#_Toc323382358)

[2.5. https://cleancodenz@github.com/cleancodenz/DrupalDev.git 4](#_Toc323382359)

[2.6. Push to remote 4](#_Toc323382360)

[2.7. How to get rid off of message box 4](#_Toc323382361)

[2.7.1. Go to repository, then .git directory, some time it is hidden 4](#_Toc323382362)

[2.7.2. In the core group 4](#_Toc323382363)

[2.7.3. Add 4](#_Toc323382364)

[3. Beautiful url 4](#_Toc323382365)

[3.1. Enable clean url first 4](#_Toc323382366)

[3.2. Install pathAuto module 5](#_Toc323382367)

[3.3. Install token module 5](#_Toc323382368)

[3.4. Administration:config:search and meta data:url alias:pattern tab 5](#_Toc323382369)

[3.5. For all previous contents, enable the automatic url 5](#_Toc323382370)

[4. Theme designing 5](#_Toc323382371)

[4.1. Folder is :sites/all/themes/mytheme 5](#_Toc323382372)

[4.2. .info file: sites/all/themes/mytheme/mytheme.info 5](#_Toc323382373)

[4.3. Defaults 5](#_Toc323382374)

[4.4. Regions in content 5](#_Toc323382375)

[4.4.1. Must have regions[content] = Content 5](#_Toc323382376)

[4.4.2. Output of region 5](#_Toc323382377)

[4.4.3. There are template (.tpl.php) files available for rendering individual blocks 5](#_Toc323382378)

[4.4.4. Order of regions definition 5](#_Toc323382379)

[4.4.5. Manually set content of a region using drupal\_set\_content('header', 'Welcome!') 6](#_Toc323382380)

[4.5. Color scheme, color module integration 6](#_Toc323382381)

[4.6. Clearing the cache: 6](#_Toc323382382)

[4.7. Template.php 6](#_Toc323382383)

[4.8. Add javascripts through behaviours 6](#_Toc323382384)

[5. Blue print 6](#_Toc323382385)

[5.1. Source file structure 6](#_Toc323382386)

[5.1.1. /blueprint/ css folder 6](#_Toc323382387)

[5.1.2. Main css files /blueprint/ 6](#_Toc323382388)

[6. Useful links 6](#_Toc323382389)

[6.1. Drupal 7 examples : 6](#_Toc323382390)

[6.2. Drupal 7 api documentation 6](#_Toc323382391)

[7. Key functions: 6](#_Toc323382392)

[8. Entiy Bundle field 13](#_Toc323382393)

[9. Node API 13](#_Toc323382394)

[10. Form API 14](#_Toc323382395)

[11. Structure (big picture) 14](#_Toc323382396)

[11.1. Content: 14](#_Toc323382397)

[11.1.1. Content type 14](#_Toc323382398)

[11.1.1.1. Edit 14](#_Toc323382399)

[11.1.1.1.1. There might be custom settings 14](#_Toc323382400)

[11.1.1.1.2. Submission form settings 14](#_Toc323382401)

[11.1.1.1.3. Publish options 14](#_Toc323382402)

[11.1.1.1.4. Display settings 14](#_Toc323382403)

[11.1.1.1.5. Comments Settings 14](#_Toc323382404)

[11.1.1.1.6. Menu Settings 14](#_Toc323382405)

[11.1.2. Manage Fields 15](#_Toc323382406)

[11.2. Menus 15](#_Toc323382407)

[11.2.1. Main menu 15](#_Toc323382408)

[11.2.2. Navigation menu 15](#_Toc323382409)

[11.2.3. Management menu 15](#_Toc323382410)

[11.2.4. User menu 15](#_Toc323382411)

[11.2.5. API 15](#_Toc323382412)

[11.2.5.1. Hook\_menu 15](#_Toc323382413)

[11.2.5.2. Navigation 16](#_Toc323382414)

[11.3. Block 16](#_Toc323382415)

[11.3.1. Pages: 16](#_Toc323382416)

[11.3.2. Content types 16](#_Toc323382417)

[11.3.3. Roles 17](#_Toc323382418)

[11.3.4. Users 17](#_Toc323382419)

[11.3.5. API 17](#_Toc323382420)

[11.3.5.1. Path: sites/all/modules/modulename 17](#_Toc323382421)

[11.3.5.2. Info file : sites/all/modules/modulename/modulename.info 17](#_Toc323382422)

[11.3.5.3. Module file : sites/all/modules/modulename/modulename.module 17](#_Toc323382423)

[11.3.5.4. Modulename\_help 17](#_Toc323382424)

[11.3.5.5. hook\_block\_info() 17](#_Toc323382425)

[11.3.5.6. hook\_menu() 17](#_Toc323382426)

[11.3.5.6.1. $items['admin/config/content/current\_posts'] 17](#_Toc323382427)

[11.4. Taxonomy 19](#_Toc323382428)

[12. People 19](#_Toc323382429)

[12.1. Role 19](#_Toc323382430)

[12.2. User 19](#_Toc323382431)

[12.3. Permission API 19](#_Toc323382432)

[12.3.1. hook\_permission 19](#_Toc323382433)

[13. Modules: Program Interface, APIs 20](#_Toc323382434)

[14. Hooks 20](#_Toc323382435)

# Database : drupal712

## Database user and password drupaluser/1234

## <url:http://localhost:81/drupal7>

## Site admin : admin/680628

# Create a git repo:

## Right click the folder, create repository

## Add files to repo

## Go on to github ,create a repo

## On desktop, create a remote with the url like

## https://cleancodenz@github.com/cleancodenz/DrupalDev.git

## Push to remote

## How to get rid off of message box

Git unlock index or continue

### Go to repository, then .git directory, some time it is hidden

### In the core group

### Add

Following :

autocrlf = input   
safecrlf = false

# Beautiful url

## Enable clean url first

It might be turned off for apache, in wamp, apache, apache modules – enable rewrite\_module

Still the clean url test will fail if the install is not using a top level domain name but a subdirectory then RewriteBase should be changed to the subdirectory name

RewriteBase /mysite

In my case /mysite is /drupal7 as my install url is <http://localhost:81/drupal7/>

## Install pathAuto module

## Install token module

## Administration:config:search and meta data:url alias:pattern tab

## For all previous contents, enable the automatic url

# Theme designing

## Folder is :sites/all/themes/mytheme

## .info file: sites/all/themes/mytheme/mytheme.info

## Defaults

These defaults apply as a group. In other words, overriding a region with regions[sub\_header] = Sub-header will omit the rest of the default regions. To gain them back, they must be redefined. This also applies to stylesheets. Even though it's not technically in a group, defining another stylesheet will prevent "style.css" from being included unless it is redefined

## Regions in content

### Must have regions[content] = Content

### Output of region

Internal names are converted into region variables inside the "page.tpl.php" template automatically. In the above example, the [header] region will output all the blocks assigned to it through the $header variable in Drupal 6, or $page['header'] in Drupal 7.

Name of region will appear on blocks management page

### There are template (.tpl.php) files available for rendering individual blocks

### Order of regions definition

The order in which the regions are defined will be reflected in the block configuration table

### Manually set content of a region using drupal\_set\_content('header', 'Welcome!')

## Color scheme, color module integration

## Clearing the cache:

admin/config/development/performance-Clear all cache to reflect the new theme changes

## Template.php

Preprocessors for generating variables before they are merged with the mark up in side .tpl.php files

The only files that are required in the theme's root is the .info file and template.php. This is cool because you can put all your .tpl.php files in a templates/ directory and Drupal will still discover them

## Add javascripts through behaviours

# Blue print

## Source file structure

### /blueprint/ css folder

### Main css files /blueprint/

Screen.css, ie.css, print.css,

/blueprint/plugin and /blueprint/src/ will be used by main css files

# Useful links

## Drupal 7 examples :

http://api.drupal.org/api/examples/7

## Drupal 7 api documentation

http://api.drupal.org/api/drupal/7

# Key functions:

* 1. Node\_view(), /modules/node/node.module

$node->

vid 6

uid 1

title First listing

log

status 1

comment 2

promote 1

sticky 0

nid 6

type cleancode\_realty

language und

created 1331669629

changed 1331669629

tnid 0

translate 0

revision\_timestamp 1331669629

revision\_uid 1

body Array [1]

und Array [1]

0 Array [5]

value This is first listing

summary

format filtered\_html

safe\_value <p>This is first listing</p>\n

safe\_summary

cleancode\_realty\_color Array [1]

cleancode\_realty\_quantity Array [1]

cleancode\_realty\_image Array [1]

und Array [1]

0 Array [13]

fid 2

alt

title

width 1024

height 768

uid 1

filename Lighthouse.jpg

uri public://Lighthouse.jpg

filemime image/jpeg

filesize 561276

status 1

timestamp 1331669629

rdf\_mapping Array [0]

rdf\_mapping Array [9]

cid 0

last\_comment\_timestamp 1331669629

last\_comment\_name <Uninitialized>

last\_comment\_uid 1

comment\_count 0

name admin

picture 0

data b:0;

* 1. node\_build\_content($node, $view\_mode = 'full', $langcode = NULL) /modules/node/node.module
     1. get node base , node\_content
     2. run node\_content\_view, on the previous base

after the fields prepare view, $node->content is

content Array [6]

body Array [16]

#theme field

#weight 1

#title Body Example Description

#access true

#label\_display hidden

#view\_mode cleancode\_realty\_list

#language und

#field\_name body

#field\_type text\_with\_summary

#field\_translatable 0

#entity\_type node

#bundle cleancode\_realty

#object stdClass

#items Array [1]

0 Array [5]

value This is first listing

summary

format filtered\_html

safe\_value <p>This is first listing</p>\n

safe\_summary

#formatter text\_summary\_or\_trimmed

0 Array [1]

cleancode\_realty\_color Array [16]

#theme field

#weight 2

#title The colors available for this object.

#access true

#label\_display hidden

#view\_mode cleancode\_realty\_list

#language und

#field\_name cleancode\_realty\_color

#field\_type text

#field\_translatable 0

#entity\_type node

#bundle cleancode\_realty

#object stdClass

#items Array [1]

0 Array [3]

value ccc

format <Uninitialized>

safe\_value ccc

#formatter cleancode\_realty\_colors

0 Array [2]

cleancode\_realty\_image Array [16]

#theme field

#weight 4

#title Upload an image:

#access true

#label\_display hidden

#view\_mode cleancode\_realty\_list

#language und

#field\_name cleancode\_realty\_image

#field\_type image

#field\_translatable 0

#entity\_type node

#bundle cleancode\_realty

#object stdClass

#items Array [1]

0 Array [13]

#formatter image

0 Array [4]

#pre\_render Array [1]

#entity\_type node

#bundle cleancode\_realty

After links added to $node->content

links Array [5]

#theme links\_\_node

#pre\_render Array [1]

0 drupal\_pre\_render\_links

#attributes Array [1]

class Array [2]

node Array [3]

#theme links\_\_node\_\_node

#links Array [0]

#attributes Array [1]

comment Array [3]

#theme links\_\_node\_\_comment

#links Array [1]

comment\_forbidden Array [2]

title <a href="/drupal7/user/login?destination=node/6%23comment-form">Log in</a> or <a href="/drupal7/user/register?destination=node/6%23comment-form">register</a> to post comments

html true

#attributes Array [1]

class Array [2]

0 links

1 inline

If there is custom view, finally the format of build which was content before

$build Array [11]

body Array [16]

cleancode\_realty\_color Array [16]

cleancode\_realty\_image Array [16]

#pre\_render Array [1]

#entity\_type node

#bundle cleancode\_realty

links Array [5]

#theme node

#node stdClass

#view\_mode cleancode\_realty\_list

#language en

* + 1. field\_attach\_prepare\_view($entity\_type, $entities, $view\_mode, $langcode = NULL)

Field structure

entity\_types Array [1]

translatable 0

settings Array [0]

storage Array [5]

foreign keys Array [1]

indexes Array [1]

id 2

field\_name body

type text\_with\_summary

module text

active 1

locked 0

cardinality 1

deleted 0

columns Array [3]

bundles Array [1]

Instance structure

$instance Array [13]

label Body Example Description

widget Array [4]

type text\_textarea\_with\_summary

settings Array [2]

rows 20

summary\_rows 5

weight 0

module text

settings Array [3]

display\_summary true

text\_processing 1

user\_register\_form false

display Array [3]

default Array [5]

label hidden

type text\_default

settings Array [0]

module text

weight 0

teaser Array [5]

label hidden

type text\_summary\_or\_trimmed

settings Array [1]

module text

weight 0

cleancode\_realty\_list Array [5]

label hidden

type text\_summary\_or\_trimmed

settings Array [1]

module text

weight 1

required false

description

length 0

default\_value <Uninitialized>

id 18

field\_id 2

field\_name body

entity\_type node

bundle cleancode\_realty

deleted 0

* + 1. s
    2. ddd
  1. The render array is

$renderable\_array Array [2]

explanation Array [1]

node\_list Array [2]

0 Array [12]

body Array [16]

cleancode\_realty\_color Array [16]

cleancode\_realty\_image Array [16]

#pre\_render Array [1]

#entity\_type node

#bundle cleancode\_realty

links Array [5]

#theme node

#node stdClass

#view\_mode cleancode\_realty\_list

#language en

#contextual\_links Array [1]

1 Array [12]

* 1. drupal\_deliver\_html\_page($page\_callback\_result), /includes/common.inc
     1. drupal\_add\_http\_header('Content-Type', 'text/html; charset=utf-8')
     2. drupal\_add\_http\_header('Content-Language', $language->language);
     3. drupal\_render\_page($page\_callback\_result);
     4. drupal\_page\_footer();
  2. drupal\_render\_page($page) , /includes/common.inc
     1. drupal\_set\_page\_content($page);

in the end $page will be like

$page Array [10]

#show\_messages true

#theme page

#theme\_wrappers Array [1]

0 html

#type page

help Array [4]

system\_help Array [4]

#sorted true

#theme\_wrappers Array [1]

#region help

content Array [4]

system\_main Array [5]

#sorted true

#theme\_wrappers Array [1]

#region content

sidebar\_first Array [5]

system\_navigation Array [6]

user\_login Array [27]

#sorted true

#theme\_wrappers Array [1]

#region sidebar\_first

footer\_firstcolumn Array [4]

current\_posts\_current\_posts Array [6]

#sorted true

#theme\_wrappers Array [1]

#region footer\_firstcolumn

footer Array [4]

system\_powered-by Array [5]

#sorted true

#theme\_wrappers Array [1]

#region footer

page\_top Array [3]

toolbar Array [3]

#theme\_wrappers Array [1]

#region page\_top

* + 1. drupal\_render($page)
    2. ddd
  1. dddd

# Entiy Bundle field

In an example of content type ‘article’

Entity-> node

Bundle-> article (subtype, or content type, or node type)

Fields -> fields attached in the bundle

# Form API

* 1. Through form constructore function $form and $formstate

$form[‘itemname’]= array(

'#type' => 'item',

'#title' => t('A form with nothing but a textfield')

)

Each of $form field will create an item on the form

$form['submit'] = array(

'#type' => 'submit',

'#value' => 'Submit',

);

A submit button

# Structure (big picture)

## Content:

### Content type

Structure->Content Type

#### Edit

##### There might be custom settings

##### Submission form settings

##### Publish options

##### Display settings

##### Comments Settings

##### Menu Settings

The menus available to place links in for this content type.

When content is created, menu tab will ask for title of the menu and where to put this new content on the menu hierachy, this is controled by the menu settings of content type

##### Main menu

Is the system one

### Manage Fields

There are three compulsary fields that come from node base type

Title, label of this can be changed when the type is created

Url path settings,

Body,

Plus custom fields

### Field API

This is used by content type api (node type api)

Providing a field requires:

\* Defining a field:

o hook\_field\_info()

o hook\_field\_schema()

o hook\_field\_validate()

o hook\_field\_is\_empty()

\* Defining a formatter for the field (the portion that outputs the field for display):

o hook\_field\_formatter\_info()

o hook\_field\_formatter\_view()

\* Defining a widget for the edit form:

o hook\_field\_widget\_info()

o hook\_field\_widget\_form()

Our module defines the field in field\_example\_field\_info(), field\_example\_field\_validate() and field\_example\_field\_is\_empty(). field\_example\_field\_schema() is implemented in field\_example.install.

Our module sets up a formatter in field\_example\_field\_formatter\_info() and field\_example\_field\_formatter\_view(). These are the API hooks that present formatted and themed output to the user.

And finally, our module defines the widet in field\_example\_field\_widget\_info() and field\_example\_field\_widget\_form(). The widget is the form element used to receive input from the user when the field is being populated.

#### field\_example\_3text\_validate($element, &$form\_state)

Validate the individual fields and then convert them into a single HTML RGB value as text.

#### hook\_field\_formatter\_info()

We need to tell Drupal that we have two different types of formatters for this field. One will change the text color, and the other will change the background color.

#### hook\_field\_formatter\_view()

Two formatters are implemented.

\* field\_example\_simple\_text just outputs markup indicating the color that was entered and uses an inline style to set the text color to that value.

\* field\_example\_color\_background does the same but also changes the background color of div.region-content

#### hook\_field\_info()

Provides the description of the field.

#### hook\_field\_is\_empty()

hook\_field\_is\_emtpy() is where Drupal asks us if this field is empty. Return TRUE if it does not contain data, FALSE if it does. This lets the form API flag an error when required fields are empty.

#### hook\_field\_schema()

Defines the database schema of the field, using the format used by the Schema API.

The data we will store here is just one 7-character element, even though the widget presents the three portions separately.

All implementations of hook\_field\_schema() must be in the module's .install file

.

#### hook\_field\_validate ()

This hook gives us a chance to validate content that's in our field. We're really only interested in the $items parameter, since it holds arrays representing content in the field we've defined. We want to verify that the items only contain RGB hex values like this: #RRGGBB. If the item validates, we do nothing. If it doesn't validate, we add our own error notification to the $errors parameter

#### hook\_field\_widget\_error()

hook\_field\_widget\_error() lets us figure out what to do with errors we might have generated in hook\_field\_validate(). Generally, we'll just call form\_error()

#### hook\_field\_widget\_form ()

hook\_widget\_form() is where Drupal tells us to create form elements for our field's widget.

We provide one of three different forms, depending on the widget type of the Form API item provided.

The 'field\_example\_colorpicker' and 'field\_example\_text' are essentially the same, but field\_example\_colorpicker adds a javascript colorpicker helper.

field\_example\_3text displays three text fields, one each for red, green, and blue. However, the field type defines a single text column, rgb, which needs an HTML color spec. Define an element validate handler that converts our r, g, and b fields into a simulated single 'rgb' form element

#### hook\_field\_widget\_info()

Three widgets are provided.

\* A simple text-only widget where the user enters the '#ffffff'.

\* A 3-textfield widget that gathers the red, green, and blue values separately.

\* A farbtastic colorpicker widget that chooses the value graphically.

These widget types will eventually show up in hook\_field\_widget\_form, where we will have to flesh them out.

#### hook\_menu()

Provides a simple user interface that tells the developer where to go

A simple page to explain to the developer what to do

\_field\_example\_page()

### Custom content type API

Structure->Content Type

Example defining a node type in code.

This is an example outlining how a module can be used to define a new node type. Our example node type will allow users to specify multiple "colors", a "quantity" and an "image" for their nodes; some kind of rudimentary inventory-tracking system, perhaps?

The basic pattern for defining a node type is to tell Drupal about the node's field types, and view modes. Drupal will then take over and manage the storage for this node type. This differs from Drupal 6, where we would have to handle all the database storage ourselves in the module.

Remember that most node types do not require any custom code, as one simply creates them using the Drupal user interface. Creating a node like this in code is a special case.

Since we only have to define our node type once, most of the code required to do this is moved to the node type's .install file. Drupal 7 has us defining most of our node structure in arrays, and passing those to node\_type\_save(). We use hook\_install() as a convenient place to define these types, and hook\_uninstall() as a convenient place to not only uninstall the data contained in these nodes, but also remove the node types from Drupal's knowledge.

In previous versions of Drupal, "teaser" and "page" were node view modes. In Drupal 7 we can define custom view modes to let the node know how it should return it's data. This module declares a custom view mode called "example\_node\_list".

#### hook\_entity\_info\_alter()

We need to modify the default node entity info by adding a new view mode to be used in functions like node\_view() or node\_build\_content().

#### hook\_field\_formatter\_info()

#### hook\_field\_formatter\_view()

@todo: We need to provide a formatter for the colors that a user is allowed to enter during node creation

#### hook\_help()

#### hook\_install()(.install)

This hook is called when the user enables the module for the first time (or on subsequent enables after the module has been uninstalled). So it's a good place to define our new node type.

We will:

\* Add the body field.

\* Configure the body field.

\* Create color, quantity, and image fields.

\* Create color, quantity, and image instances.

See also

node\_type\_set\_defaults()

field\_info\_instance()

field\_update\_instance()

field\_create\_field()

field\_create\_instance()

#### hook\_menu()

We are providing a default page to illustrate the use of our custom node view mode that will live at <http://example.com/?q=examples/node_example>

node\_example\_page()

Callback that builds our content and returns it to the browser.

This callback comes from hook\_menu().

Return value

a build array a renderable array showing a list of our nodes.

See also

node\_load()

node\_view()

node\_example\_field\_formatter\_view()

#### hook\_theme()

This lets us tell Drupal about our theme functions and their arguments.

A custom theme function.

theme\_example\_node\_color($variables)

By using this function to format our node-specific information, themes can override this presentation if they wish. This is a simplifed theme function purely for illustrative purposes

#### hook\_uninstall(.install)

This hook is called when the user not only has disabled the module, but also uninstalls it from the 'uninstall' tab in the module page.

So it's a perfect time to remove our fields and instances and new node type from the database.

\_node\_example\_installed\_fields()

Returns a structured array defining the fields created by this content type.

This is factored into this function so it can be used in both node\_example\_install() and node\_example\_uninstall().

Return value

An associative array specifying the fields we wish to add to our new node type.

Related topics

\_node\_example\_installed\_instances()

Returns a structured array defining the instances for this content type.

The instance lets Drupal know which widget to use to allow the user to enter data and how to react in different view modes. We are going to display a page that uses a custom "node\_example\_list" view mode. We will set a cardinality of three allowing our content type to give the user three color fields.

This is factored into this function so it can be used in both node\_example\_install() and node\_example\_uninstall().

Return value

An associative array specifying the instances we wish to add to our new node type.

### Custom field type API(Node API)

What node api does is create independent fields that can be enabled to attach to an existing content type

Hook\_form\_alter will attach the new fields to existing editor form and configuration in node type edit form to allow administrator to config.

It is different from a new node type which has its own defined fields

#### Hook\_form\_alter

By implementing this hook, we're able to modify any form. We'll only make changes to two types: a node's content type configuration and edit forms.

We need to have a way for administrators to indicate which content types should have our rating field added. This is done by inserting radios in the node's content type configuration page.

Changes made by this hook will be shown when editing the settings of any content type.

Optionally, hook\_form\_FORM\_ID\_alter() could be used with the function name nodeapi\_example\_form\_node\_type\_form\_alter

#### Hook\_node\_delete

When a node is deleted, we need to remove all related records from our table, including all revisions. For the delete operations we use node->nid

#### Hook\_node\_insert

As a new node is being inserted into the database, we need to do our own database inserts

#### Hook\_node\_insert

Loads the rating information if available for any of the nodes in the argument list.

#### Hook\_node\_update

As an existing node is being updated in the database, we need to do our own database updates.

This hook is called when an existing node has been changed. We can't simply update, since the node may not have a rating saved, thus no database field. So we first check the database for a rating. If there is one, we update it. Otherwise, we call nodeapi\_example\_node\_insert() to create one

#### Hook\_node\_validate

Check that the rating attribute is set in the form submission, since the field is required. If not, send error message

#### Hook\_node\_view

This is a typical implementation that simply runs the node text through the output filters.

Finally, we need to take care of displaying our rating when the node is viewed. This operation is called after the node has already been prepared into HTML and filtered as necessary, so we know we are dealing with an HTML teaser and body. We will inject our additional information at the front of the node copy.

Using node API 'hook\_node\_view' is more appropriate than using a filter here, because filters transform user-supplied content, whereas we are extending it with additional information

#### Hook\_schema(.install)

#### Hook\_uninstall(.install)

We need to clean up our variables data when uninstalling our module.

Our implementation of nodeapi\_example\_form\_alter() automatically creates a nodeapi\_example\_node\_type\_<contentType> variable for each node type the user wants to rate.

To delete our variables we call variable\_del for our variables' namespace, 'nodeapi\_example\_node\_type\_'. Note that an average module would have known variables that it had created, and it could just delete those explicitly. For example, see render\_example\_uninstall(). It's important not to delete variables that might be owned by other modules, so normally we would just explicitly delete a set of known variables.

hook\_uninstall() will only be called when uninstalling a module, not when disabling a module. This allows our data to stay in the database if the user only disables our module without uninstalling it.

#### Hook\_theme

#### Custom theme function

theme\_nodeapi\_example\_rating($variables)

A custom theme function.

By using this function to format our rating, themes can override this presentation if they wish; for example, they could provide a star graphic for the rating. We also wrap the default presentation in a CSS class that is prefixed by the module name. This way, style sheets can modify the output without requiring theme code.

## Menus

### Main menu

This is the system one, some time due to theme, it might not be expandable for children nodes.

But then you view children nodes, the navigation crumbs will always be right.

In some themes the main menu can be hard coded, that is without enabling the main menu on any block, you can still can see main menu, like the default theme.

### Navigation menu

This is the system one too

Expandable children can be displayed

### Management menu

This is the system one

### User menu

This is the system one

### API

#### Hook\_menu

"access callback" with an optional "access arguments" of each menu item is called before the page callback proceeds

Type:

MENU\_NORMAL\_ITEM

MENU\_DEFAULT\_LOCAL\_TASK and MENU\_LOCAL\_TASK will appear as tabs on its parent url /parent/localtask1

Will appear as tabs on page /parent/

The default task normally is not linked to its provided path, but to its parent instead.

page callback should be a function, or a general one like : drupal\_get\_form

The menu will form the links into a tree structure according its route

A

a/b

a/c

#### Navigation

Navigation is a block which can be placed in the structure-blocks

Menus defined in hook\_menu will be added to Navigation automatically

In the menus, new menu blocks like Navigation can be added, within this menu block you can add other links to this menu block, the links must be provided, it is not selected

## Block

Blocks are placed in regions which come from theme

A block can be installed as module with block type

A menu is a block, so it can be placed in regions

### Pages:

The block can be placed on only certain pages

### Content types

Only show on certain content types

### Roles

### Users

### API

#### Path: sites/all/modules/modulename

#### Info file : sites/all/modules/modulename/modulename.info

#### Module file : sites/all/modules/modulename/modulename.module

#### Modulename\_help

At admin module list, the help link is provided

#### hook\_block\_info()

Provided the human name for this block, the name will appear in the admin modules list

#### hook\_menu()

##### $items['admin/config/content/current\_posts']

adds a config entry for this block on Admin/Configuration page

###### Title :

The title of configuration page, the title of link on the admin configs page

###### Description:

The description of the link on the admin configs page

###### page callback

The function call if the link is clicked, here is drupal\_get\_form

###### page arguments

The arguments of page call back

Array first item is the form implementation function name

###### Form implementation of above setting

$form['current\_posts\_max'] = array(

'#type' => 'textfield',

'#title' => t('Maximum number of posts'),

'#default\_value' => variable\_get('current\_posts\_max', 3),

'#size' => 2,

'#maxlength' => 2,

'#description' => t('The maximum number of links to display in the block.'),

'#required' => TRUE,

);

###### Form validation for form implementation

$form\_state['values']['current\_posts\_max']; the way to access the variable on form

###### access arguments

Which access control to use hook\_permission()

?????

###### type

MENU\_NORMAL\_ITEM,

Menu type -- A "normal" menu item that's shown in menu and breadcrumbs.

##### $items[modulename]

###### Type

MENU\_CALLBACK

Menu type -- A hidden, internal callback, typically used for API calls

###### Page call back page implementation

Set the title of page

drupal\_set\_title('Johnson current posts11111111');

Get the contents, it is an object or array

Set the theme of items

$page\_array['current\_posts\_arguments'] = array(

'#title' => t('All posts from the last week'),

'#items' => $items,

//Theme hook with suggestion.

'#theme' => 'item\_list\_\_current\_posts',

#### hook\_block\_view()

This is the implementation of view of block normally with a more link to the page

##### set the title of block

$block['subject'] = t('Johnson current posts22222222');

##### get the contents

Set the theme of items

//Pass data through theme function.

$block['content']['posts'] = array(

'#theme' => 'item\_list\_\_current\_posts\_\_block',

'#items' => $items,

);

//Add a link to the page for more entries.

$block['content']['more'] = array(

'#theme' => 'more\_link\_\_current\_posts',

'#url' => 'current\_posts',

'#title' => t('See the full list of current posts.'),

);

## Taxonomy

# People

## Role

Roles created to group users and to link with permissions.

### Create role

Admin – people-permissions- roles tab

## User

### Admin-configuration-account settings

To change basic settings of user management, like the how many fields are used by user profile.

## Permission

Permission are assigned to roles, and permission is checked when it is required to access or operate in drupal.

For any content type, there are 5 system permissions that can be used by drupal, either through ui or through permission api

#### Admin-people-permission

## Permission API

### hook\_permission

return array(

'create cc real estate listing' =>

array('title' => t('Create access to real estate listing contents')),

'edit any cc real estate listing' =>

array('title' => t('Edit access to any real estate listings')),

'edit own cc real estate listings'=>

array('title' => t('Edit access to their own real estate listing contents')),

'delete any cc real estate listing' =>

array('title' => t('Delete access to any real estate listings')),

'delete own cc real estate listings'=>

array('title' => t('Delete access to their own real estate listing contents')),

);

This will actually create new permissions for this module

### hook\_access

This applies or checks the custom defined permission in hook\_permission with actions

[php]

if ($op == 'create') {

// Only users with permission to do so may create this node type.

return user\_access('create cc real estate listing', $account);

}

[/php]

# Modules: Program Interface, APIs

## Path: sites/all/modules

## Hooks

### hook\_help($path, $arg)

path could either be a registered path in hook\_menu

using ‘%’ for wildcard matching

arg can provide further params

To provide a help page for a whole module with a listing on admin/help, your hook implementation should match a path with a special descriptor after a "#" sign: 'admin/help#modulename' The main module help text, displayed on the admin/help/modulename page and linked to from the admin/help page

### hook\_install

This hook is called when the user enables the module for the first time

(or on subsequent enables after the module has been uninstalled).

So it's a good place to define our new node type

### hook\_uninstall

This hook is called when the user not only has disabled the module but also uninstalls it from the 'uninstall' tab in the module page

Only modules that implemented uninstall appear on uninstall tab

# Eclipse configuration for drupal model development

## Install hook template to enable code completion

## Download the templates from the Eclipse Hook Templates project page

## In Eclipse, go to Preferences > PHP > Editor > Templates and 'Import' the downloaded XML file. Apply your changes

### Use the templates by typing the name of the hook (eg. 'hook\_form') and pressing ctrl-space to pop up the auto-complete window.

### http://drupal.org/project/eclipse

## File extension support

Under Window -> Preferences (Eclipse

-> Preferences on Mac) menu:

\* Expand the left-hand menu to General -> Content Types. Under Content types on the right, click Text -> PHP Content Type. Add the \*.engine, \*.theme, \*.install, \*.inc, \*.module \*.profile and \*.test file types so that these files will be recognized as PHP. The free plug-in to Eclipse for Drupal developers does this for you and is easy to install.

Under Window -> Preferences (Eclipse

-> Preferences on Mac) menu:

\* In the left-hand menu click on General -> Workspace: Check Text file encoding and select

Other :

UTF-8

Check New text file line delimiter and select

Other:

Unix

Under Window -> Preferences (Eclipse

-> Preferences on Mac) menu:

\* In the left-hand menu click on General -> Editors -> Text

Editors: Check Insert spaces for tabs.

In the left-hand menu click on PHP -> Code Style ->

Formatter: Set Tab policy: Spaces. Set Indentation size to 2.

# dddd