The average content management system (CMS) is like a toy truck—specific assumptions have been made about how it will be used, and these assumptions are difficult to override.

Drupal is designed to be the perfect content management solution for non-technical users who needs both simplicity and flexibility. Content management frameworks, on the other hand, are like the raw materials needed to make any toy—no assumptions have been made about how they’ll be used, rather a collection of wheels, windshields, axles, frames, etc., that a toy maker can easily connect together. With Drupal, a maker could create a toy truck, but she or he could just as easily create a toy airplane, submarine, or robot.

ou want the homepage to have a section featuring the five most recent ones.

In other ordinary cms u vl need to download plugins but in case of drupal u can very achieve this by using views.

**Node**

Essentially, a node is a set of related bits of information. When you create a new blog post, you are not only defining its body text, but also its title, content, author link, creation date, taxonomy (tags), etc.

A node is the generic term for a piece of content on your web site. The content type of the node will define what fields are included with it. Depending on the type of node, different fields will be attached, and this is known as a content type. For example, a basic Pagecontent type has attached fields such as title and body fields. Other examples of content type are: Book pages for use in Books, Discussion topics in forums, Blog pages in blogs, and News articles.

**Drupal Flow**

1.To display anything u need to have some data to be shown this is at the base of the system and collection of nodes

2. The next layer up is where modules live. Modules are functional plugins that are either part of the Drupal core (they ship with Drupal) or they are contributed items that have been created by members of the Drupal community.

3. t the next layer, we find blocks and menus. Blocks often provide the output from a module or can be created to display whatever you want, and then can be placed in various spots (Regions) in your template (theme) layout. Blocks can be configured to output in various ways, as well as only showing on certain defined pages, or only for certain defined users.

**Menus** are a core element of Drupal. Menus are a navigator of Drupal which provide links to all the pages created in Drupal.

4. Next are user permissions. This is where settings are configured to determine what different kinds of users are allowed to do and see. Permissions are defined for various roles, and in turn, users are assigned to these roles in order to grant them the defined permissions.

5. On the top layer is the site theme (the "skin"). This is made up predominantly of XHTML and CSS, with some PHP variables intermixed, so Drupal-generated content can go in the appropriate spots.

* **Custom Functionality:** Are you frustrated with web site builders or limitations of Wordpress? Drupal may be the answer. Drupal is highly customizable and even allows the integration of web applications and mashups using third party APIs.
* **Flexible Implementation:** Drupal allows your website to evolve in any direction. For example, you may start with a blog but then want the option of adding other features like a wiki, electronic commerce, forums, multiple content types, etc.
* **Complex Components:** Complex forms, workflows, multilingual sites, and multi-sites can be set up easily with Drupal. There are also over 16,000 Drupal modules available to provide excellent functionality for your website, from commerce to directories and image galleries.
* **Configurable Website:** Drupal can easily be configured to interact with other sites or technologies. If you want, you can even configure it to interact with another Drupal site!
* **Customize-able Content Types:** With Drupal, you have ability to create your own content types. For example, you can create a custom format for directory listings or (CCK module prebuild in drupal7 etc). This is one of the most used features of Drupal.
* **List, Sort, & Search Information:**  
  You can quickly [organize and display lists of information](https://www.drupal.org/project/views).

**Entity types**

An *entity type* is a useful abstraction to group together fields. Entity types are used to store and display data, which can be nodes (content), comments, taxonomy terms, user profiles, or something custom developed.

Read more about Entities in the [Entity API](http://drupal.org/node/1261744) documentation.

**Comment**

Comments are another type of content you can have on your site (if you have enabled the core Comment module). Each comment is typically a small piece of content that a user submits, attached to a particular node. For example, each piece of discussion attached to a particular forum topic node is a comment.

**Taxonomy**

Drupal has a system for classifying content known as *taxonomy*. This is provided by the core Taxonomy module. You can define your own *vocabularies* (groups of *taxonomy terms*) and add terms to each vocabulary. Each vocabulary can then be attached to one or more content types, and in this way, nodes on your site can be grouped into categories, tagged, or classified in any way you choose.

Read more about this concept in the [taxonomy module](http://drupal.org/documentation/modules/taxonomy) documentation.

**User**

A user is a type of entity which represents a real-world website user. By default, a user has a set of properties including their username, password, role, and e-mail address. However, they may also have other properties provided by other modules, and can be extended with new fields. For example, you could add a new "Link" field for a user's Twitter address.

**Module**

A module is software (code) that extends Drupal functionality. Modules fall into one of three categories:

* **Core** modules are those included with the main download of Drupal. These can be turned on or off without downloading additional components. Examples include *Blog, Book, Poll*, or *Taxonomy*.
* **Contributed** modules are downloaded from the Modules download section of drupal.org, and installed within your Drupal installation. Examples include *Panels, Views*or *Metatag*.
* **Custom** modules are modules you write yourself. This requires a thorough understanding of Drupal, PHP programming, and Drupal's API.

For a collection of useful materials about module development, see [Module Development with Drupal](https://www.drupal.org/module-development).

**Regions & Blocks**

Pages on your Drupal site are laid out in *Regions*. These can include the header, footer, sidebars,featured top,featured bottom, main content regions etc.. . Your theme may define additional regions.

*Blocks* are discrete chunks of information that are displayed in the regions of your site's pages. Blocks can take the form of static chunks of HTML or text, menus (which are for site navigation)

, the output from modules (e.g. hot forum topics), or dynamic listings that you've created yourself (e.g. a list of upcoming events).

**Menus**

There are four standard menus in Drupal 7:

* The **Main** menu is built by site administrators and displayed automatically in the page header of many themes (and if not, you can enable their blocks to display them).
* **Management** is the administration menu, and is presented in the Admin toolbar.
* **Navigation** is a catch-all menu that usually contains links supplied by modules on your site.
* **User** menu contains links to the User account and the logout link.

You can also create your own custom menus, and display them by enabling their blocks.

You can customize menus in several ways, such as reordering menu items by setting their “weight” or simply dragging them into place, renaming menu items, and changing the link title (the tooltip that appears when you mouse over a menu item). You can move a menu item into a different menu by editing the Parent property of the menu item.

You can also add custom menu items to a menu, from the Add menu item tab of the Menu administration screen. To create a menu item, you will need to provide the path to the content.

In all cases a menu item will only be shown to a visitor if they have the rights to view the page it links to. For example, the admin menu item is not shown to visitors who are not logged in.

**Theme**

The *Theme layer* is separate from the data layer, the functionality extension layer (module) and Core. Theme controls the appearance (look and feel) of your site, or how your site is displayed, including the graphic look, layout, and colors. A theme consists of one or more PHP template files that define the HTML output of your site's pages, along with one or more CSS files that define the layout, fonts, colors, and other styles.

For a collection of useful materials for themers, see [Theming and Front End Development with Drupal](https://www.drupal.org/theming).

**Views**

Although not all sites have Views, most sites include the Views module because of the excellent tools it provides. Views allows people to choose a list of nodes or other entities and present them as pages, blocks, RSS feeds, or other formats. The main use case for views is to create dynamically updating lists of content (for example, a listing of latest news), based on properties of that content (in the case of the news listing, that the content type is “News” and sorted by publication date).

**Drupal as MVC**

[**http://www.garfieldtech.cobvbm/blog/mvc-vs-pac**](http://www.garfieldtech.cobvbm/blog/mvc-vs-pac)

### MVC

The most commonly-known interactive system architecture is [Model-View-Controller](http://en.wikipedia.org/wiki/Model-view-controller), or MVC. Most good desktop applications use MVC or a variant of it, sometimes with the Controller partially merged into the View. In MVC, as the pretty picture at the other end of that links shows, the Model holds data, the View is the part the user sees, and the Controller is an intermediary for business logic. Seems reasonable, right? Now take a closer look.

### PAC

A less publicized but still widely used architecture is [Presentation-Abstraction-Control](http://en.wikipedia.org/wiki/Presentation-abstraction-control), or PAC. The two main differences between MVC and PAC are that in PAC the Presentation component is "dumb" while all the intelligence resides in the Controller and PAC is layered. Again, see the pretty picture.

You'll notice that the Presentation and Abstraction components never speak to each other. The Controller takes input, not the display component. The Controller has all the business logic and routing information. The Presentation component is essentially just a filter that takes raw data that the Controller pushes through it and renders it to HTML (or WML, or XML, or text, or an icon in a graphical monitoring system, or whatever). It's just a templating system.

Drupal

 Drupal is very much a PAC architecture. In fact, it's a rather good PAC architecture. The menu system acts as the Controller. It accepts input via a single source (HTTP GET and POST), routes requests to the appropriate helper functions, pulls data out of the Abstraction (nodes and, in Drupal 5, forms), and then pushes it through a filter to get a Presentation of it (the theme system). .

Drupal bootstrapping

Drupal architecture

Hierarchy mvc

Currently the closest Drupal comes to MVC is the [Panels module](http://drupal.org/project/panels). Panels allows the site admin to set up a custom layout with custom regions and then pull blocks, Views, or nodes into it in those regions. Currently that's the only real pull-based logic that Drupal supports for display given the active discouragement of database access (even via accessors like node\_load()) in theme\_functions and template files. Even that, however, is still limited to specific panel pages. There's no supported way to randomly pull a block into a node page, for instance. I know Earl "merlin" Miles is deeply engrossed in Panels 2.0, The Next Generation, but I don't believe it currently involves turning Panels inside out. He is, of course, welcome to correct me if he does. ;-)

Menu depth 9

Db count unlimited

Php 5.2

Staging db connect

$databases = array(

'default' =>

array(

'default' =>

array(

'database' => 'db\_expert\_search\_new',

'username' => 'root',

'password' => '',

'host' => 'localhost',

'port' => '',

'driver' => 'mysql',

'prefix' => '',

),

),

'db1' => array(

'default' =>

array(

'database' => 'db\_expert\_search\_new',

'username' => 'root',

'password' => '',

'host' => 'localhost',

'port' => '',

'driver' => 'mysql',

'prefix' => '',

),

),

);

Robots no sites- because alt title metatag ..prevents crawling

Htaccess : change url of old sie

Append www to datamatics

[www.dmatics.com/Index.php](http://www.dmatics.com/Index.php)

Taxonomy

User

Custom menu

# **Users, permissions, and roles**

Last updated May 30, 2016. Created on March 10, 2013.Edited by[*davidjguru*](https://www.drupal.org/u/davidjguru),[*avinashm*](https://www.drupal.org/u/avinashm),[*gdaw*](https://www.drupal.org/u/gdaw),[*chriscerk*](https://www.drupal.org/u/chriscerk).[*Log in to edit this page*](https://www.drupal.org/user?destination=node/1938536).

Every visitor to your site, whether they have an account and log in or visit the site anonymously, is considered a user to Drupal. Every user also has a numeric user ID special to the type of user.

## **Types of Users**

### Master Administrator

This user has the ID one (1). User of ID one (1) is the primary admin user account created during Drupal installation. This user is very special because it has permission to do absolutely everything on the site.

### Logged In

These users are assigned a user ID when they register for the website. A user name and email address is associated with any user that isn't anonymous (therefore must be logged in).

### Anonymous

Anonymous users who visit the website but do not login all share a user ID of zero (0).

#### Associating more information with Users

Other information can also be associated with users by modules; for instance, if you use the core Profile module, you can define user profile fields to be associated with each user.

## **Permissions**

Other users on your site can be assigned permissions via roles. To do this, you first need to create a role by navigating to people --> permissions --> role. A common role is "Content editor" or "Member". Next, you will assign permissions to that role, to tell Drupal what that role can and can't do on the site. Finally, you will grant certain users on your site your new role, which will mean that when those users are logged in, Drupal will let them do the actions you gave that role permission to do.

You can also assign permissions for the special built-in roles of "anonymous user" (a user who is not logged in) and "authenticated user" (a user who is logged in, with no special role assignments). Drupal permissions are quite flexible—you are allowed to assign permission for any task to any role, depending

drupal features -- terms my exp

reuasibilty --

**FEATURE**

modules

security - not easy to break, informs us

it makes seasy to create n manage

translates anything using the site with built in translate ui

an open source no licensicng cost

social media contents publish

**What is drupal**

cms, open php, manage organise pulish,non technical simplicity flexibilty, raw material blocks, menus and .. integrate it together and create any page as per ur require,ment

**CORE DRUPAL compo**

modules menu taxonomy,poll, theme

**NODE**

is a entity

entity

entity type 4 commnts nodes users taxo

feature

patch

glossary

**CDN**

This module provides easy Content Delivery Network integration for Drupal sites. It changes file URLs, so that files (CSS, JS, images, fonts, videos …) are downloaded from a CDN instead of your web server.

assets can be cached in a proxy which is geographically closer to the end user, which usually leads to lower latency and increased download speed;

Each page response is shared between your origin server and the CDN, meaning that your origin server can serve more concurrent requests;

**Apache solr**

Solr (pronounced "solar") is an open source enterprise search platform, written in Java, from the Apache Lucene project.

Its major features include full-text search, hit highlighting, faceted search, real-time indexing, dynamic clustering, database integration, NoSQL features[1] and rich document (e.g., Word, PDF) handling.

Providing distributed search and index replication, Solr is designed for scalability and fault tolerance.[2]

Solr is the second-most popular enterprise search engine after Elasticsearch.

However, when the number of documents to search is potentially large, or the quantity of search queries to perform is substantial, the problem of full-text search is often divided into two tasks: indexing and searching. The indexing stage will scan the text of all the documents and build a list of search terms (often called an index, but more correctly named a concordance). In the search stage, when performing a specific query, only the index is referenced, rather than the text of the original documents.[2]

The indexer will make an entry in the index for each term or word found in a document, and possibly note its relative position within the document. Usually the indexer will ignore stop words (such as "the" and "and") that are both common and insufficiently meaningful to be useful in searching.

**OOPS start**

**Class**

A class is the core of any modern Object Oriented Programming language such as C#.

In OOP languages it is mandatory to create a class for representing data.

A class is a blueprint of an object that contains variables for storing data and functions to perform operations on the data.

A class will not occupy any memory space

This is a programmer-defined data type

You can think of a class as a template for making many instances of the same kind (or class) of object.

**Object**

"An object is an instance of a class"

**Abstraction**

Abstraction is "To represent the essential feature without representing the background details."

Abstraction lets you focus on what the object does instead of how it does it.

Abstraction is the process of hiding the working style of an object, and showing the information of an object in an understandable manner.

cannot b instantitated

abstract class MobilePhone

{

public void Calling();

public void SendSMS();

}

public class Nokia1400 : MobilePhone

{

}

public class Nokia2700 : MobilePhone

{

public void FMRadio();

public void MP3();

public void Camera();

}

public class BlackBerry : MobilePhone

{

public void FMRadio();

public void MP3();

public void Camera();

public void Recording();

public void ReadAndSendEmails();

}

Abstraction is a common thing.

**Encapsulation**

Encapsulation is a technique used to protect the information in an object from another object.

Hide the data for security such as making the variables private, and expose the property to access the private data that will be public.

Example 1

class Demo

{

private int \_mark;

public int Mark

{

get { return \_mark; }

set { if (\_mark > 0) \_mark = value; else \_mark = 0; }

}

}

Phone the user only learns how to use the Mobile Phone but not how the Mobile Phone works.

**Inheritance**

When a class includes a property of another class it is known as inheritance.

Inheritance is a process of object reusability.

For example, a child includes the properties of its parents.

public class ParentClass

{

public ParentClass()

{

Console.WriteLine("Parent Constructor.");

}

public void print()

{

Console.WriteLine("I'm a Parent Class.");

}

}

public class ChildClass : ParentClass

{

public ChildClass()

{

Console.WriteLine("Child Constructor.");

}

public static void Main()

{

ChildClass child = new ChildClass();

child.print();

}

}

Output

Parent Constructor.

Child Constructor.

I'm a Parent Class.

**Parent class** − A class that is inherited from by another class. This is also called a base class or super class.

**Child Class** − A class that inherits from another class. This is also called a subclass or derived class.

**Polymorphism**

Polymorphism means one name, many forms.

One function behaves in different forms.

function name will remain same but it make take different number of arguments and can do different task.

**Interface**

Any method you create in the interface can not contain any concrete code inside of it.

**Static**

Declaring class members or methods as static makes them accessible without needing an instantiation of the class.

<?php

class Foo {

public static $my\_static = 'foo';

public function staticValue() {

return self::$my\_static;

}

}

print Foo::$my\_static . "\n";

$foo = new Foo();

print $foo->staticValue() . "\n";

?>

**Final Keyword**

PHP 5 introduces the final keyword, which prevents child classes from overriding a method by prefixing the definition with final.

**Diff**

Multiple and multilevel both type of inheritance is possible in the interface. But single and multilevel inheritance is possible in abstract classes.

In abstract classes, this is not necessary that every method should be abstract and regular fucntions with definition. But in interface every method is abstract and only declared.

The method of PHP interface must be public only. A method in an abstract class in PHP could be public or protected both.

**OOPS end**

**Features start**

The features module enables the capture and management of features in Drupal. A feature is a collection of Drupal entities which taken together satisfy a certain use-case.

Features provides a UI and API for taking different site building components from modules with exportables and bundling them together in a single feature module. A feature module is like any other Drupal module except that it contains additional information in its info file so that configuration can be checked, updated, or reverted programmatically.

*If u create multiple content types, related taxo, related views, related panel page variant u can easily get the same thing on other site or pantheon - by simple creating a feature*

*Creating a feature as sson as u select content type it will take all related taxo dependency fields.. views panel page doenload feature and place sites modules folder and enable it on ur site*

*Structure admin oage shows default*

The Features module will notice any changes made to a component of any features and signal this with an Overridden state on the Features administration page.

### "Revert" vs "Update"

Use one of these operations if the configuration on your site (living in the database) differs from the definitions in your feature module (living in code).

**Revert**  
This operation changes your site configuration (living in the database) to match up with the definitions in the feature module code.

In some cases (e.g. views), the revert operation will delete the respective configuration in the database, so the system will instead use the default configuration that is defined in the feature module's code.

**Update / Recreate**

The update operation will produce a modified version of your feature module, which matches up with the configuration found in the database.

What happens with this modified code, depends on whether you do this with drush or with the web-based interface.

*diagram*

If you update the feature via drush ("drush features-update [name]"), the modified code will replace your existing feature module in the filesystem.

The "recreate" button in the web-based administration interface is the equivalent to "update". However, the modified code will be provided for download as a .tar archive, instead of overwriting the existing feature module. It is up to the site builder to manually unzip and re-upload the modified module, replacing the original.

**Locking**

The 2.x branch of Features introduced a UI for "locking" either individual component types of a given feature or the feature as a whole by clicking on a lock icon. Clicking the icon again will unlock the feature or component type.

While component types can be locked in a given feature, there is no way of locking individual items. For example, you can lock all views in feature\_x, but can't lock only a single view.

A feature or component that is "locked" may show as overridden but will not revert, even if features revert is called. Thus, locking is a simple way to protect specific changes from being lost when features are reverted.

**Coding for unfpa charts**

$path = drupal\_get\_path('module', 'unfpa\_global\_dashboard');

$theme\_path = drupal\_get\_path('theme', 'unfpa\_global');

drupal\_add\_css($path . '/css/dashboard.css');

drupal\_add\_js($path . '/js/html2canvas.js');

$file\_path = drupal\_realpath('public://') . '/adolscent-youth/files/config\_files/data\_v46/mapdatafiles/mapData.json';

// This function is used to read the json contents of a file.

$map\_data = file\_get\_contents($file\_path, true);

// This function is used to get proper content for json decoding.

$json\_data = str\_replace(array('mapdata\_callback(', ')'), "", $map\_data);

$ay\_data = array();

$ay\_data = json\_decode($json\_data, true);

drupal\_add\_js(array('unfpa\_global\_data' => $country, 'dashboard\_name' => $dashboard\_name), 'setting');

drupal\_add\_js(drupal\_get\_path('theme', 'unfpa\_global') . '/templates/html5worldmap/worldmap.js');

simplemaps\_worldmap\_mapdata.state\_specific = Drupal.settings.mapdata[selOpt];

simlemaps\_worldmap.load();

simplemaps\_worldmap.hooks.complete = function () {

simplemaps\_worldmap.region\_zoom('0');

**Features end**

**Drupal code issue start**

array

fetchobject

db\_query n select better

LIKE case

->fields('t', array('tid', 'name')) why array?

keyup keydown

selector

function custom\_header\_search\_form($form, &$form\_state) why 3 param

stylesheets[all]

document .ready n windiw . load

function $ when to use y to use

value input

==============

echo drupal\_json\_encode();

node - nid title

taxonomy\_term\_name tid name vid

taxonomy\_vocabulary vid name

->condition('title','%'.$string.'%','LIKE')

function search\_block\_data() {

return "<input type='text' id='searchbox' placeholder='search by expert or area of expertise' autocomplete='off'/><div id='matches' class='scrollbox'></div>";

}

(function($){ /\*\*\*/ })(jQuery);

$(document).ready(function () { /\*\*\*/ });

$this.keyup(function(){ /\*\*\*/ });

===============

$block['content'] = drupal\_get\_form('custom\_header\_search\_form');

function custom\_header\_search\_form($form, &$form\_state) {

// Build your form

$options[0] = 'Site';

$options[1] = 'Library';

$options[2] = 'Database';

$form['opt'] = array(

'#type' => 'select',

'#options' => $options,

'#prefix' => '<div class="poll-form select-dropdown">',

'#suffix' => '</div>',

);

function custom\_header\_search\_form\_submit($form, &$form\_state) {

$search = $form['opt']['#options'][$form\_state['values']['opt']];

$search\_value = $form\_state['values']['searchbox'];

$form\_state['redirect'] = url($alias);

================

$form['#submit'][] = 'expertsearch\_general\_settings\_form\_submit';

return system\_settings\_form($form);

===========

custom map

$query = db\_select('field\_data\_field\_maplocation', 'f');

$query->join('node', 'n', 'n.nid = f.entity\_id');

$query->fields('f', array('field\_maplocation\_value'));

$query->condition('status', 1, '=');

$result = $query->execute();

array\_unique

function custom\_map\_preprocess\_page(&$variables) {

function custom\_map\_views\_pre\_render(&$view) {

location\_array = <?php echo json\_encode($locations); ?>;</script>

later

---

function custom\_map\_views\_pre\_render(&$view) {

if ($view->name == "map\_location\_experts") {

$key = $view->args[0];

$field = field\_info\_field('field\_maplocation');

$output = isset($field['settings']['allowed\_values'][$key]) ? $field['settings']['allowed\_values'][$key] : '';

$view->set\_title($output);

}

}------------

====================================================

flood table

=======================

features

deploy

----

ravi will explain

metadata

**Drupal code issue end**

**Multisite steps start**

**Multisite steps end**

**Flexslider steps start**

1. **Create custom block from structure blocks**

<div class="flexslider">

<ul class="slides">

<li>Hello</li>

<li>flex</li>

<li>check</li>

</ul>

</div>

1. **Add script.js**

(function ($) {

$(document).ready(function () {

$window.load(function () {

$('.flexslider').flexslider({

animation: "slide",

animationLoop: true,

itemWidth: 200,

itemMargin: 15,

controlNav: true,

minItems: getGridSize(), // use function to pull in initial value

maxItems: getGridSize(), // use function to pull in initial value

start: function (slider) {

flexslider = slider;

}

});

});

// check grid size on resize event

$window.resize(function () {

//$(".front .flexslider ul.slides").width('100%');

var gridSize = getGridSize();

flexslider.vars.minItems = gridSize;

flexslider.vars.maxItems = gridSize;

});

});

})(jQuery);

1. **Add flexslider.js**

In theme folders sites/all/themes/urtheme/js

1. **Add flexslider.css**

In theme folders sites/all/themes/urtheme/css

**Flexslider steps end**

**Varnish steps start**

**Varnish steps end**

**Multilingual steps start**

**Multilingual steps end**

**Multidomain steps start**

1. **Domain Access module dwnld and install**

enable all except "**Domain strict"**

1. **Create sites.php**

add a line

**$sites['expert.americanancestors.org'] = 'default';**

**$sites['expertsearch-test.org'] = 'default';**

1. **Add a line in httpd-vhosts.conf**

<VirtualHost \*:80>

ServerAdmin webmaster@dummy-host2.example.com

DocumentRoot "D:/project/htdocs/expertsearchbackup"

ServerName expert.americanancestors.org

ErrorLog "logs/expert.americanancestors.org-error.log"

CustomLog "logs/expert.americanancestors.org-access.log" common

<Directory "D:/project/htdocs/expertsearch">

Order allow,deny

Allow from all

</Directory>

</VirtualHost>

<VirtualHost \*:80>

ServerAdmin webmaster@dummy-host2.example.com

DocumentRoot "D:/project/htdocs/expertsearchbackup"

ServerName expertsearch-test.org

ErrorLog "logs/expertsearch-test.org-error.log"

CustomLog "logs/expertsearch-test.org-access.log" common

<Directory "D:/project/htdocs/expertsearch">

Order allow,deny

Allow from all

</Directory>

</VirtualHost>

<VirtualHost \*:80>

ServerAdmin webmaster@dummy-host2.example.com

DocumentRoot "D:/project/htdocs"

ServerName localhost

ErrorLog "logs/localhost-error.log"

CustomLog "logs/localhost-access.log" common

<Directory "D:/project/htdocs/expertsearch">

Order allow,deny

Allow from all

</Directory>

</VirtualHost>

1. **Add a line in settings.php**

/\*\*

\* Add the domain module setup routine.

\*/

include DRUPAL\_ROOT . '/sites/all/modules/contrib/domain/settings.inc';

1. **Go to structure-> domain -> create domain paste the alias nd see the domain in domain list**
2. **After above steps see the links**

**main url :** [**http://expert.americanancestors.org/**](http://expert.americanancestors.org/)

**domain :** [**http://expertsearch-test.org/**](http://expertsearch-test.org/)

**Multidomain steps end**

**Feed steps start**

**Feed steps end**

**Code ay map start**

$path = drupal\_get\_path('module', 'unfpa\_global\_dashboard');

$theme\_path = drupal\_get\_path('theme', 'unfpa\_global');

drupal\_add\_css($path . '/css/dashboard.css');

drupal\_add\_js($path . '/js/html2canvas.js');

$file\_path = drupal\_realpath('public://') . '/adolscent-youth/files/config\_files/data\_v46/mapdatafiles/mapData.json';

// This function is used to read the json contents of a file.

$map\_data = file\_get\_contents($file\_path, true);

// This function is used to get proper content for json decoding.

$json\_data = str\_replace(array('mapdata\_callback(', ')'), "", $map\_data);

$ay\_data = array();

$ay\_data = json\_decode($json\_data, true);

drupal\_add\_js(array('unfpa\_global\_data' => $country, 'dashboard\_name' => $dashboard\_name), 'setting');

drupal\_add\_js(drupal\_get\_path('theme', 'unfpa\_global') . '/templates/html5worldmap/worldmap.js');

simplemaps\_worldmap\_mapdata.state\_specific = Drupal.settings.mapdata[selOpt];

simlemaps\_worldmap.load();

simplemaps\_worldmap.hooks.complete = function () {

simplemaps\_worldmap.region\_zoom('0');

**Code ay map end**

**Views tpl php start**

**hook\_preprocess\_views\_view(**

/\*\*

\* Implements hook\_preprocess\_views\_view().

\*/

function unfpa\_global\_executive\_board\_preprocess\_views\_view(&$variables) {

if ($variables['view']->name != 'executive\_board') {

return;

}

if (!in\_array($variables['view']->current\_display, array('bureau\_members'))) {

return;

}

$variables['year'] = isset($variables['view']->exposed\_raw\_input['field\_eb\_time\_of\_mandate\_value']['value']) ? $variables['view']->exposed\_raw\_input['field\_eb\_time\_of\_mandate\_value']['value'] : date("Y");

}

**hook\_views\_pre\_render**

/\*\*

\* Implements hook\_views\_pre\_render().

\*/

function unfpa\_global\_executive\_board\_views\_pre\_render(&$view) {

if ($view->name != 'executive\_board') {

return;

}

if (!in\_array($view->current\_display, array('bureau\_members'))) {

return;

}

$result=$view->result;

foreach($result as $row){

$field\_field\_eb\_time\_of\_mandate = $row->field\_field\_eb\_time\_of\_mandate[0]['rendered']['#markup'];

if ($field\_field\_eb\_time\_of\_mandate) {

$row->field\_field\_eb\_time\_of\_mandate[0]['rendered']['#markup'] = t("Time of mandate") . ": (" . str\_replace("to", "-", $field\_field\_eb\_time\_of\_mandate) . ")";

}

}

}

**Views-view--executive-board—bureau-members.tpl.php (views-view-viewmachinename—displayname.tpl.php)**

<?php

/\*\*

\* @file

\* Main view template.

\*/

?>

<div class="<?php print $classes; ?>">

<div class="exbo-title-social-wrapper clearfix">

<h1 class="page-title executive-board"><?php print t("Members of the Executive Board"); ?></h1>

</div>

<div class="exec-brd view-events-header">

<span class="selected-one"><?php print t('Select one:'); ?></span>

<div class="select-list">

<a class="bluelink" href="<?php echo "board-members"; ?>"><?php print t('Board Members'); ?></a>

<a class="active bluelink bureau-members" href= "<?php echo "bureau-members"; ?>"><?php print t('Bureau Members'); ?></a>

</div>

<div class="clearfix"></div>

</div>

<?php if ($header): ?>

<div class="exec-brd view-header">

<?php print $header; ?>

</div>

<?php endif; ?>

<?php if ($exposed): ?>

<div class="exec-brd-exposed-filter-container">

<div class="filter-fields exec-brd-filter-text"><?php

print t('Bureau Members');

?>

<?php if ($year): ?>

<span><?php print $year; ?></span>

<?php endif; ?>

</div>

<div class="filter-fields exec-brd-view-filters">

<div class="view-filters">

<?php print $exposed; ?>

</div>

</div>

</div>

<?php endif; ?>

<?php if ($attachment\_before): ?>

<div class="attachment attachment-before">

<?php print $attachment\_before; ?>

</div>

<?php endif; ?>

<?php if ($rows): ?>

<div class="view-content">

<?php print $rows; ?>

</div>

<?php elseif ($empty): ?>

<div class="view-empty">

<?php print $empty; ?>

</div>

<?php endif; ?>

<?php print render($title\_prefix); ?>

<?php if ($title): ?>

<?php print $title; ?>

<?php endif; ?>

<?php print render($title\_suffix); ?>

<?php if ($pager): ?>

<?php print $pager; ?>

<?php endif; ?>

<?php if ($attachment\_after): ?>

<div class="attachment attachment-after">

<?php print $attachment\_after; ?>

</div>

<?php endif; ?>

<?php if ($more): ?>

<?php print $more; ?>

<?php endif; ?>

<?php if ($footer): ?>

<div class="view-footer">

<?php print $footer; ?>

</div>

<?php endif; ?>

<?php if ($feed\_icon): ?>

<div class="feed-icon">

<?php print $feed\_icon; ?>

</div>

<?php endif; ?>

</div><?php /\* class view \*/ ?>

**Views-view-fields-vw-results—home-results-panel.tpl.php (views-view-fields—viewname--displayname)**

<?php

/\*\*

\* @file

\* This template is used to print a single field in a view.

\*

\* It is not actually used in default Views, as this is registered as a theme

\* function which has better performance. For single overrides, the template is

\* perfectly okay.

\*

\* Variables available:

\* - $view: The view object

\* - $field: The field handler object that can process the input

\* - $row: The raw SQL result that can be used

\* - $output: The processed output that will normally be used.

\*

\* When fetching output from the $row, this construct should be used:

\* $data = $row->{$field->field\_alias}

\*

\* The above will guarantee that you'll always get the correct data,

\* regardless of any changes in the aliasing that might happen if

\* the view is modified.

\*/

?>

<?php if (isset($row->field\_field\_related\_topic[0]['raw']['entity']->path['alias'])): ?>

<a href="<?php echo $row->field\_field\_related\_topic[0]['raw']['entity']->path['alias']; ?>" title="click here to learn more about '<?php echo $row->field\_field\_related\_topic[0]['raw']['entity']->title; ?>'">

<h3 class="result\_title"><?php echo $row->field\_field\_first\_title[0]['raw']['value']; ?></h3>

<div class="result\_logo\_icon">

<img typeof="foaf:Image" src="<?php echo file\_create\_url($row->field\_field\_logo\_icon[0]['raw']['uri']); ?>" width="30" height="30" alt="">

</div>

<div class="value\_wrapper">

<div class="value" style="color:<?php echo $row->field\_field\_color\_code[0]['raw']['value']; ?>;"><?php echo $row->field\_field\_result\_number[0]['raw']['value']; ?></div>

<div class="result\_sub\_title"><?php echo $row->field\_field\_sub\_title\_result[0]['raw']['value']; ?>

<?php if (isset($row->field\_field\_source[0]['raw']['value'])): ?>

<span class="star">\*</span>

<span class="arrow"></span>

<p class="text">

<?php echo $row->field\_field\_source[0]['raw']['value']; ?>

</p>

<?php endif; ?>

</div>

</div>

</a>

<?php else : ?>

<h3 class="result\_title"><?php echo $row->field\_field\_first\_title[0]['raw']['value']; ?></h3>

<div class="result\_logo\_icon">

<img typeof="foaf:Image" src="<?php echo file\_create\_url($row->field\_field\_logo\_icon[0]['raw']['uri']); ?>" width="30" height="30" alt="">

</div>

<div class="value\_wrapper">

<div class="value" style="color:<?php echo $row->field\_field\_color\_code[0]['raw']['value']; ?>;"><?php echo $row->field\_field\_result\_number[0]['raw']['value']; ?></div>

<div class="result\_sub\_title"><?php echo $row->field\_field\_sub\_title\_result[0]['raw']['value']; ?>

<?php if (isset($row->field\_field\_source[0]['raw']['value'])): ?>

<span class="star">\*</span>

<span class="arrow"></span>

<p class="text">

<?php echo $row->field\_field\_source[0]['raw']['value']; ?>

</p>

<?php endif; ?>

</div>

</div>

<?php endif; ?>

**Views-view-field—viewname--displayname—fieldname.tpl.php**

<?php

/\*\*

\* @file

\* This template is used to print a single field in a view.

\*

\* It is not actually used in default Views, as this is registered as a theme

\* function which has better performance. For single overrides, the template is

\* perfectly okay.

\*

\* Variables available:

\* - $view: The view object

\* - $field: The field handler object that can process the input

\* - $row: The raw SQL result that can be used

\* - $output: The processed output that will normally be used.

\*

\* When fetching output from the $row, this construct should be used:

\* $data = $row->{$field->field\_alias}

\*

\* The above will guarantee that you'll always get the correct data,

\* regardless of any changes in the aliasing that might happen if

\* the view is modified.

\*/

?>

<?php

$title = '';

if(!empty($output))

{

switch($output)

{

case '#FF0000':

$title = "Unsatisfactory";

break;

case '#FFFF00':

$title = "Partially Satisfactory";

break;

case '#008000':

$title = "Satisfactory";

break;

}

}

?>

<div class="audit-rating-color" style="background: <?php print $output; ?>" title="<?php echo $title; ?>"></div>

**Views tpl php end**

**Hook menu start**

Define menu items and page callbacks.

This hook enables modules to register paths in order to define how URL requests are handled. Paths may be registered for URL handling only, or they can register a link to be placed in a menu (usually the Navigation menu). A path and its associated information is commonly called a "menu router item". This hook is rarely called (for example, when modules are enabled), and its results are cached in the database.

[hook\_menu](https://api.drupal.org/api/drupal/modules%21system%21system.api.php/function/hook_menu/7.x)() implementations return an associative array whose keys define paths and whose values are an associative array of properties for each path. (The complete list of properties is in the return value section below.)

### Callback Functions

The definition for each path may include a page callback function, which is invoked when the registered path is requested. If there is no other registered path that fits the requested path better, any further path components are passed to the callback function. For example, your module could register path 'abc/def':

function mymodule\_menu() {

$items['abc/def'] = array(

'page callback' => 'mymodule\_abc\_view',

);

return $items;

}

function mymodule\_abc\_view($ghi = 0, $jkl = '') {

// ...

}

When path 'abc/def' is requested, no further path components are in the request, and no additional arguments are passed to the callback function (so $ghi and $jkl would take the default values as defined in the function signature). When 'abc/def/123/foo' is requested, $ghi will be '123' and $jkl will be 'foo'. Note that this automatic passing of optional path arguments applies only to page and theme callback functions.

#### Callback Arguments

In addition to optional path arguments, the page callback and other callback functions may specify argument lists as arrays. These argument lists may contain both fixed/hard-coded argument values and integers that correspond to path components. When integers are used and the callback function is called, the corresponding path components will be substituted for the integers. That is, the integer 0 in an argument list will be replaced with the first path component, integer 1 with the second, and so on (path components are numbered starting from zero). To pass an integer without it being replaced with its respective path component, use the string value of the integer (e.g., '1') as the argument value. This substitution feature allows you to re-use a callback function for several different paths. For example:

function mymodule\_menu() {

$items['abc/def'] = array(

'page callback' => 'mymodule\_abc\_view',

'page arguments' => array(1, 'foo'),

);

return $items;

}

When path 'abc/def' is requested, the page callback function will get 'def' as the first argument and (always) 'foo' as the second argument.

If a page callback function uses an argument list array, and its path is requested with optional path arguments, then the list array's arguments are passed to the callback function first, followed by the optional path arguments. Using the above example, when path 'abc/def/bar/baz' is requested, mymodule\_abc\_view() will be called with 'def', 'foo', 'bar' and 'baz' as arguments, in that order.

Special care should be taken for the page callback [drupal\_get\_form](https://api.drupal.org/api/drupal/includes%21form.inc/function/drupal_get_form/7.x)(), because your specific form callback function will always receive $form and &$form\_state as the first function arguments:

function mymodule\_abc\_form($form, &$form\_state) {

// ...

return $form;

}

See [Form API documentation](https://api.drupal.org/api/drupal/includes%21form.inc/group/form_api/7.x) for details.

### Wildcards in Paths

#### Simple Wildcards

Wildcards within paths also work with integer substitution. For example, your module could register path 'my-module/%/edit':

$items['my-module/%/edit'] = array(

'page callback' => 'mymodule\_abc\_edit',

'page arguments' => array(1),

);

When path 'my-module/foo/edit' is requested, integer 1 will be replaced with 'foo' and passed to the callback function. Note that wildcards may not be used as the first component.

#### Auto-Loader Wildcards

Registered paths may also contain special "auto-loader" wildcard components in the form of '%mymodule\_abc', where the '%' part means that this path component is a wildcard, and the 'mymodule\_abc' part defines the prefix for a load function, which here would be named mymodule\_abc\_load(). When a matching path is requested, your load function will receive as its first argument the path component in the position of the wildcard; load functions may also be passed additional arguments (see "load arguments" in the return value section below). For example, your module could register path 'my-module/%mymodule\_abc/edit':

$items['my-module/%mymodule\_abc/edit'] = array(

'page callback' => 'mymodule\_abc\_edit',

'page arguments' => array(1),

);

When path 'my-module/123/edit' is requested, your load function mymodule\_abc\_load() will be invoked with the argument '123', and should load and return an "abc" object with internal id 123:

function mymodule\_abc\_load($abc\_id) {

return [**db\_query**](https://api.drupal.org/api/drupal/includes%21database%21database.inc/function/db_query/7.x)("SELECT \* FROM {mymodule\_abc} WHERE abc\_id = :abc\_id", array(':abc\_id' => $abc\_id))->[**fetchObject**](https://api.drupal.org/api/drupal/7.x/search/fetchObject)();

}

This 'abc' object will then be passed into the callback functions defined for the menu item, such as the page callback function mymodule\_abc\_edit() to replace the integer 1 in the argument array. Note that a load function should return FALSE when it is unable to provide a loadable object. For example, the [node\_load](https://api.drupal.org/api/drupal/modules%21node%21node.module/function/node_load/7.x)() function for the 'node/%node/edit' menu item will return FALSE for the path 'node/999/edit' if a node with a node ID of 999 does not exist. The menu routing system will return a 404 error in this case.

### endering Menu Items As Tabs

You can also make groups of menu items to be rendered (by default) as tabs on a page. To do that, first create one menu item of type [MENU\_NORMAL\_ITEM](https://api.drupal.org/api/drupal/includes%21menu.inc/constant/MENU_NORMAL_ITEM/7.x), with your chosen path, such as 'foo'. Then duplicate that menu item, using a subdirectory path, such as 'foo/tab1', and changing the type to [MENU\_DEFAULT\_LOCAL\_TASK](https://api.drupal.org/api/drupal/includes%21menu.inc/constant/MENU_DEFAULT_LOCAL_TASK/7.x) to make it the default tab for the group. Then add the additional tab items, with paths such as "foo/tab2" etc., with type[MENU\_LOCAL\_TASK](https://api.drupal.org/api/drupal/includes%21menu.inc/constant/MENU_LOCAL_TASK/7.x). Example:

// Make "Foo settings" appear on the admin Config page

$items['admin/config/system/foo'] = array(

'title' => 'Foo settings',

'type' => [**MENU\_NORMAL\_ITEM**](https://api.drupal.org/api/drupal/includes%21menu.inc/constant/MENU_NORMAL_ITEM/7.x),

// Page callback, etc. need to be added here.

);

// Make "Tab 1" the main tab on the "Foo settings" page

$items['admin/config/system/foo/tab1'] = array(

'title' => 'Tab 1',

'type' => [**MENU\_DEFAULT\_LOCAL\_TASK**](https://api.drupal.org/api/drupal/includes%21menu.inc/constant/MENU_DEFAULT_LOCAL_TASK/7.x),

// Access callback, page callback, and theme callback will be inherited

// from 'admin/config/system/foo', if not specified here to override.

);

// Make an additional tab called "Tab 2" on "Foo settings"

$items['admin/config/system/foo/tab2'] = array(

'title' => 'Tab 2',

'type' => [**MENU\_LOCAL\_TASK**](https://api.drupal.org/api/drupal/includes%21menu.inc/constant/MENU_LOCAL_TASK/7.x),

// Page callback and theme callback will be inherited from

// 'admin/config/system/foo', if not specified here to override.

// Need to add access callback or access arguments.

);

### Return value

An array of menu items. Each menu item has a key corresponding to the Drupal path being registered. The corresponding array value is an associative array that may contain the following key-value pairs:

* **"title"**: Required. The untranslated title of the menu item.
* **"title callback"**: Function to generate the title; defaults to [t](https://api.drupal.org/api/drupal/includes%21bootstrap.inc/function/t/7.x)(). If you require only the raw string to be output, set this to FALSE.
* **"title arguments"**: Arguments to send to [t](https://api.drupal.org/api/drupal/includes%21bootstrap.inc/function/t/7.x)() or your custom callback, with path component substitution as described above.
* **"description"**: The untranslated description of the menu item.
* **"page callback"**: The function to call to display a web page when the user visits the path. If omitted, the parent menu item's callback will be used instead.
* **"page arguments"**: An array of arguments to pass to the page callback function, with path component substitution as described above.
* **"delivery callback"**: The function to call to package the result of the page callback function and send it to the browser. Defaults to [drupal\_deliver\_html\_page](https://api.drupal.org/api/drupal/includes%21common.inc/function/drupal_deliver_html_page/7.x)() unless a value is inherited from a parent menu item. Note that this function is called even if the access checks fail, so any custom delivery callback function should take that into account. See [drupal\_deliver\_html\_page](https://api.drupal.org/api/drupal/includes%21common.inc/function/drupal_deliver_html_page/7.x)() for an example.
* **"access callback"**: A function returning TRUE if the user has access rights to this menu item, and FALSE if not. It can also be a boolean constant instead of a function, and you can also use numeric values (will be cast to boolean). Defaults to [user\_access](https://api.drupal.org/api/drupal/modules%21user%21user.module/function/user_access/7.x)() unless a value is inherited from the parent menu item; only [MENU\_DEFAULT\_LOCAL\_TASK](https://api.drupal.org/api/drupal/includes%21menu.inc/constant/MENU_DEFAULT_LOCAL_TASK/7.x) items can inherit access callbacks. To use the [user\_access](https://api.drupal.org/api/drupal/modules%21user%21user.module/function/user_access/7.x)() default callback, you must specify the permission to check as 'access arguments' (see below).
* **"access arguments"**: An array of arguments to pass to the access callback function, with path component substitution as described above. If the access callback is inherited (see above), the access arguments will be inherited with it, unless overridden in the child menu item.
* **"theme callback"**: (optional) A function returning the machine-readable name of the theme that will be used to render the page. If not provided, the value will be inherited from a parent menu item. If there is no theme callback, or if the function does not return the name of a current active theme on the site, the theme for this page will be determined by either [hook\_custom\_theme](https://api.drupal.org/api/drupal/modules%21system%21system.api.php/function/hook_custom_theme/7.x)() or the default theme instead. As a general rule, the use of theme callback functions should be limited to pages whose functionality is very closely tied to a particular theme, since they can only be overridden by modules which specifically target those pages in [hook\_menu\_alter](https://api.drupal.org/api/drupal/modules%21system%21system.api.php/function/hook_menu_alter/7.x)(). Modules implementing more generic theme switching functionality (for example, a module which allows the theme to be set dynamically based on the current user's role) should use[hook\_custom\_theme](https://api.drupal.org/api/drupal/modules%21system%21system.api.php/function/hook_custom_theme/7.x)() instead.
* **"theme arguments"**: An array of arguments to pass to the theme callback function, with path component substitution as described above.
* **"file"**: A file that will be included before the page callback is called; this allows page callback functions to be in separate files. The file should be relative to the implementing module's directory unless otherwise specified by the "file path" option. Does not apply to other callbacks (only page callback).
* **"file path"**: The path to the directory containing the file specified in "file". This defaults to the path to the module implementing the hook.
* **"load arguments"**: An array of arguments to be passed to each of the wildcard object loaders in the path, after the path argument itself. For example, if a module registers path node/%node/revisions/%/view with load arguments set to [array](http://php.net/array)(3), the '%node' in the path indicates that the loader function [node\_load](https://api.drupal.org/api/drupal/modules%21node%21node.module/function/node_load/7.x)() will be called with the second path component as the first argument. The 3 in the load arguments indicates that the fourth path component will also be passed to [node\_load](https://api.drupal.org/api/drupal/modules%21node%21node.module/function/node_load/7.x)() (numbering of path components starts at zero). So, if path node/12/revisions/29/view is requested, [node\_load](https://api.drupal.org/api/drupal/modules%21node%21node.module/function/node_load/7.x)(12, 29) will be called. There are also two "magic" values that can be used in load arguments. "%index" indicates the index of the wildcard path component. "%map" indicates the path components as an array. For example, if a module registers for several paths of the form 'user/%user\_category/edit/\*', all of them can use the same load function [user\_category\_load](https://api.drupal.org/api/drupal/modules%21user%21user.module/function/user_category_load/7.x)(), by setting the load arguments to [array](http://php.net/array)('%map', '%index'). For instance, if the user is editing category 'foo' by requesting path 'user/32/edit/foo', the load function [user\_category\_load](https://api.drupal.org/api/drupal/modules%21user%21user.module/function/user_category_load/7.x)() will be called with 32 as its first argument, the array ('user', 32, 'edit', 'foo') as the map argument, and 1 as the index argument (because %user\_category is the second path component and numbering starts at zero). [user\_category\_load](https://api.drupal.org/api/drupal/modules%21user%21user.module/function/user_category_load/7.x)() can then use these values to extract the information that 'foo' is the category being requested.
* **"weight"**: An integer that determines the relative position of items in the menu; higher-weighted items sink. Defaults to 0. Menu items with the same weight are ordered alphabetically.
* **"menu\_name"**: Optional. Set this to a custom menu if you don't want your item to be placed in Navigation.
* **"expanded"**: Optional. If set to TRUE, and if a menu link is provided for this menu item (as a result of other properties), then the menu link is always expanded, equivalent to its 'always expanded' checkbox being set in the UI.

**Hook menu end**