Índice de contenido

API de Drupal

Una API (del inglés *Application Programming Interface*) es un interfaz para comunicar componentes de un sistema de software. Para cada componente del sistema una API maneja la comunicación entre el componente y el núcleo. Este método hace que los cambios en el componente se aislen de los correspondientes cambios en el núcleo, de este modo, se reducen los esfuerzos en depuración y pruebas a los componentes únicamente.

En general, una API establece un estándar para tratar con operaciones de bajo nivel e

introducir estabilidad y uniformidad en el código. El ejemplo más común de una API es la API de la base de datos, que encapsula las operaciones de la base de datos desde el núcleo, de tal modo que el núcleo funciona sin tener en cuenta el gestor de base de datos que el sistema usa.

Ahora se verán unos ejemplos de cómo podemos usar la API de Drupal.

1. db_query

Con db_query es posible hacer consultas select a la base de datos. Ahora se verán las siguientes líneas de código:

```
<?php
$uid = 1;
$result = db_query('SELECT n.nid, n.title, n.created
FROM {node} n WHERE n.uid = %d', $uid);
while ($r = db_fetch_array($result)) {
  $rows[] = $r;
}
print theme('table', $header, $rows);
?>
```

Primero se fija el valor del argumento ($uid=1). Seguidamente se ejecuta la consulta (se recogen valores de la tabla node cuyo argumento es el uid). Y finalmente, se recorren mediante db_fetch_array los resultados de la consulta imprimiendo los valores mediante el layout de tabla.

La llamada a la API de db_query es la siguiente:

db_query($query)

Parámetros

$query Una cadena de texto conteniendo una consulta SQL.

... Un número variable de argumento que son sustituidos dentro de la consulta usan sintaxis printf(). En vez de un número de variables es posible pasar un solo array conteniendo los argumentos de la consulta.

Modificadores válidos son: %s, %d, %f, %b (datos binarios) y %%.

Valor devuelto

Un recurso resultado de consulta a base de datos, ó FALSE si la consulta no fue ejecutada correctamente.

Es posible consultar la API de db_query, para una descripción completa de la función y sus argumentos.

2. db_fetch_array

db_fetch_array($result)
Captura el resultado de una fila desde la anterior consulta como un array.

Parámetros

$result Un recurso resultado de consulta a base de datos, lo que devuelve db_query().

Valor devuelto

Un array asociativo representando la siguiente fila del resultado o FALSE. Las claves de este objeto son los nombres de los campos de la tabla seleccionada por la consulta y los valores del objeto son los valores de la tabla para esa fila.

3. node_load

En general, para obtener un nodo desde un nid es más cómodo utilizar node_load que carga un objeto nodo desde la base de datos. Ahora se verá un ejemplo de código:

```
$nid = 1;
$node = node_load($nid);
print($node->title);
```

Primero se fija el valor del argumento ($nid=1). Seguidamente se carga el nodo cuyo nid es $nid en $node. Y, finalmente, se imprime el título del nodo o lo que se quiera.

La llamada a la API de node_load es la siguiente:

node_load($param = array(), $revision = NULL, $reset = NULL)

Carga un objeto nodo desde la base de datos.

Parámetros

$param O bien el nid del nodo ó un array de condiciones con las que hacer la consulta en la base de datos

$revision Which numbered revision to load. Defaults to the current version.

$reset Si resetear la cache interna node_load.

Valor devuelto

Un objeto nodo completamente poblado.

Es posible consultar la API de node_load, para una descripción completa de la función y sus argumentos.

4. user_load

De la misma manera, que no se suele usar db_query para cargar nodos, tampoco se usa para cargar usuarios. user_load carga un objeto usuario de la base de datos. Ahora se verá un ejemplo de código:

```
$uid = 1;
$account = user_load($uid);

print($account->name);
```

Primero se fija el valor del argumento ($uid=1). Seguidamente se carga el usuario cuyo uid es $uid en la tabla users. Y, finalmente, se imprime el nombre del usuario o lo que se necesite.

La llamada a la API de node_load es la siguiente:

user_load($user_info = array())

Carga un objeto usuario

Parámetros

$user_info Información acerca del usuario a cargar, consistiendo en uno de los siguientes:

- Un array asociativo cuyas claves son los campos en la tabla {users} table (tales como uid, name, pass, mail, status) y cuyos valores son los valores de los campos
- Un user ID numérico.

Valor devuelto

Un objeto usuario completamente cargado si ha habido éxito en la carga, si el usuario no pudo ser cargado devuelve FALSE.

Es posible consultar la API de user_load, para una descripción completa de la función y sus argumentos.

5. Aplicando funciones de la API a un proyecto Drupal

Ahora se mostrará cómo aplicar las funciones aprendidas a un proyecto Drupal. Se puede hacer desde varias partes:

- Customfield php de views
- Templates del tema instalado
- Templates de módulos instalados
- Templates que modifican views
- Módulos nuevos
- Desde contenidos ó bloques con el filtro php activo

La parte de nuevos módulos se abordará en el próximo capítulo, pero el resto de hacks que se pueden hacer a Drupal sí se va a mostrar desde aquí.

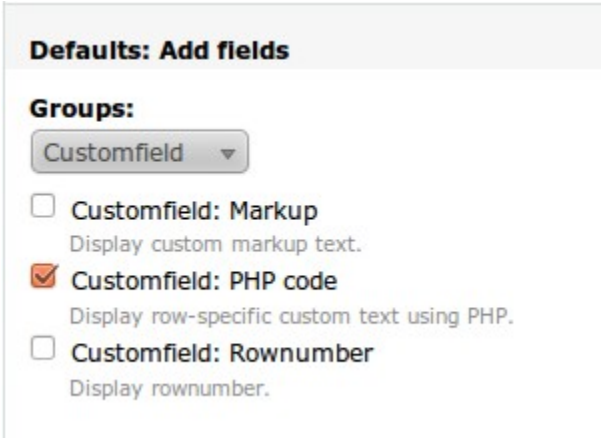
5.1 Views Customfield Php

Para tener esta funcionalidad es necesario tener instalado el módulo customfield php. Este módulo te permite insertar código php en un campo de views. Se verá cómo hacerlo

con un ejemplo.

Dada la siguiente vista:

Se pulsa en + en el apartado “Fields”. Va a encontrarse con un desplegable de categorías de fields y se selecciona “Customfield”



Y ahí se elige “Customfield: PHP Code” y pulsamos “Add”.

Ahora es el momento de recordar user_load y ver que la variable disponible $data->users_uid puede servir como valor de entrada. De este modo, se introduce el siguiente código y se pulsa en actualizar:

```
$account = user_load($data->users_uid);                    return $account->name;
```

Luego es necesario salvar la vista, teniendo un display de página ó de bloque.

Este ejemplo podría (y debe) ser fácilmente implementado usando views. Pero ilustra muy bien el poder de customfield php.

5.2 Templates del tema instalado

Otra situación en la que se puede querer aplicar los conocimientos de la API. Es modificando algún template del tema que está activo ó que se ha desarrollado para el proyecto actual.

Es posible que por alguna razón un usuario pida que después del contenido de un nodo aparezca el nombre del administrador del sitio (cuyo uid es 1). En el caso de que el tema activo fuera garland se abriría el siguiente fichero: “themes/garland/templates/node.tpl.php” y debajo de print $content; se introducirían las líneas de código relativas a la función user_load que ya se han visto. Quedando todo del siguiente modo:

```
...
  <div class="content clear-block">
    <?php print $content ?>
  </div>
<?php
$uid = 1;
$account = user_load($uid);
print($account->name);
?>
   <div class="clear-block">
```

```
...
```

5.3 Templates de módulos instalados

Modificar templates de módulos instalados es similar a lo visto en el anterior apartado. Tan solo es necesario localizar el template del módulo que se pretende modificar, copiarlo en la carpeta del tema y hacer allí las modificaciones pertinentes.

5.4 Templates que modifican views

Si desde una vista se pulsa a Basic settings → Theme se ven los diferentes nombres con los que es posible reescribir las plantillas relacionadas con views. Una vez elegida la plantilla a reescribir, se introduce el fichero en la carpeta del tema.

Ahora que ya se sabe donde aplicar la API, se puede seguir aprendiendo nuevas funciones.

6. drupal_set_message

Imprime un mensaje, normalmente con la acción que se acaba de realizar. Un ejemplo:

```
<?php
drupal_set_message('Aprendiendo a usar drupal_set_message');
?>
```

Este código hace aparecer la frase "Aprendiendo a usar drupal_set_message" formateado como mensaje Drupal, normalmente dentro de una caja verde.

La llamada a la API de drupal_set_message es la siguiente:

```
drupal_set_message($message = NULL, $type = 'status', $repeat = TRUE)
```

Parámetros

$message El mensaje empieza con una mayúscula y finaliza con un punto.

$type El tipo de mensaje. Uno de los siguientes valores es posible:

- 'status'
- 'warning'
- 'error'

$repeat Si esto es FALSE y el mensaje ha sido asignado, entonces el mensaje no será repetido.

Es posible consultar la API de drupal_set_message, para una descripción completa de la función y sus argumentos.

7. node_save

Graba un objeto nodo en la base de datos ó añade un nuevo nodo. Ahora se verá un ejemplo de código:

```
<?php
$node = node_load(1); // load node
$node->title = 'nah, nah, nah, nah, boo, boo'; // Do some changes
```

```
node_save($node); // save your changes
drupal_set_message('Updated');
?>
```

En el ejemplo se puebla un objeto nodo cuyo nid es 1. Se establece un título cualquiera. Se guarda el nodo modificado en la base de datos. Y finalmente se imprime el mensaje “Updated”.

La llamada a la API de node_save es la siguiente:

```
node_save(&$node)
```

Es posible consultar la API de node_save, para una descripción completa de la función y sus argumentos.

8. user_save

De la misma manera que se escriben las modificaciones a un nodo en base de datos, es posible escribir en base de datos las modificaciones a un usuario. Se presentan un par de ejemplos:

```
<?php
$account = user_load(1); // load node
user_save($account, array('name' => 'John Smith')); // save your changes
drupal_set_message('Updated');
?>
```

Este ejemplo es bastante similar al de node_save: tras cargar el usuario, las modificaciones se realizan al nombre del usuario y se salvan los datos. Ahora el siguiente ejemplo:

```
<?php
$details = array(
   'name' => 'John Smith',
   'pass' => 'sssh es un secreto',
   'mail' => 'john@smith.com',
   'access' => 0, /* optional, i didnt want user to be able to this this account yet*/
   'status' => 1
);
$user = user_save(null, $details);
?>
```

En este ejemplo se usa user_save para crear usuarios nuevos, que es otro de los usos de esta función, los valores se definen en el array $details.

La llamada a la API de user_save es la siguiente:

```
user_save($account, $array = array(), $category = 'account')
```

Parámetros

$account El objeto de usuario para modificar ó añadir. Si $user->uid es omitido, se añadirá un nuevo usuario.

$edit Un array de campos y valores a guardar. Por ejemplo, array('name' => 'My name'). Asignando NULL a un campo, borra el valor en la base de datos.

$category (optional) La categoría para almacenar información del perfil.

Valor devuelto

Un objeto $user si hay éxito y FALSE en caso contrario.

Es posible consultar la API de user_save, para una descripción completa de la función y sus argumentos.

9. watchdog

Hace un log de un mensaje de sistema. Ahora se verá un ejemplo de código:

```
<?php
function node_save_action($node) {
  node_save($node);
  watchdog('action', 'Saved @type %title', array('@type' => node_get_types('name', $node), '%title' =>
$node->title));
}
?>
```

Antes de explicar el código, se explicará la declaración de la API:

watchdog($type, $message, $variables = array(), $severity = WATCHDOG_NOTICE, $link = NULL)

Parámetros

$type La categoría al que el mensaje pertenece. Puede ser cualquier cadena, pero la práctica general es usar el nombre del módulo llamando watchdog().

$message El mensaje a almacenar en el log. ¡Deja el $message traducible no concatenando valores dinámicos dentro de él!

$variables Array de variables para reemplazar en el mensaje ó NULL si el mensaje ya está traducido ó no es posible traducirlo

$severity La gravedad del mensaje, dada en el RFC 3164. Posibles valores son WATCHDOG_ERROR, WATCHDOG_WARNING, etc.

$link Un enlace asociado con el mensaje.

En el ejemplo $type es “action”, $message es “Saved @type %tittle” y las variables type y title se obtienen mediante el array.

Es posible consultar la API de watchdog, para una descripción completa de la función y sus argumentos.

10. t

La función t facilita la traducción de cadenas en Drupal. Ahora se verá un ejemplo de código:

```
<?php
print(t('Hello World'));
?>
```

Este código imprime la cadena “Hello world”. Pero ahora la cadena “Hello world” es traducible. Si se va a Administración → Construcción del Sitio → Traducir Interfaz → Buscar se puede buscar la cadena “Hello world” y traducirla por “Hola Mundo” en español.

Ahora se verá otro ejemplo:

```
$uid = 1;

$account = user_load($uid);

print(t("@name's blog", array('@name'=> $account->name)));
```

Como se ve es posible introducir variables en la cadena a traducir, éstas son definidas en el segundo argumento de t que es un array.
La llamada a la API de t es la siguiente:

```
t($string, $args = array(), $langcode = NULL)
```

Es posible consultar la API de t, para una descripción completa de la función y sus argumentos.

11. Ejercicios

Evalúe como verdadero ó falso las siguientes afirmaciones:

1. Existe una única manera de extraer valores de un nodo de la base de datos.
2. drupal_set_message escribe en el log de Drupal.
3. node.tpl.php es un fichero donde podemos probar la api de Drupal.
4. node.tpl.php es un fichero donde podemos implementar hook de Drupal.
5. Para cargar un usuario con user_load basta con conocer algún campo de la tabla users.
6. watchdog no escribe en la base de datos de Drupal.
7. db_fectch_array requiere haber utilizado db_query previamente.
8. Con node_save() es posible crear nodos nuevos, pero con user_save() no es posible crear usuarios nuevos.
9. Es posible usar php dentro de views sin necesidad de software adicional.

10. El valor devuelto por user_load() es un array.

Desarrollo de Módulos

1. Arquitectura de los Módulos de Drupal

¿Qué es exactamente un módulo y cuál es su propósito?

La segunda pregunta es fácil de responder: un módulo Drupal es un mecanismo diseñado para proporcionar un método uniforme de extender las funcionalidades de Drupal. Esta respuesta acerca bastante a responder a la primera cuestión. Un módulo es un conjunto de código PHP y ficheros que usan la API de Drupal y la arquitectura para integrar nuevos componentes funcionales dentro del framework Drupal. Un propósito de este manual es explicar cómo escribir este código.

Se empezará ahora a echar un vistazo a la arquitectura del módulo. Los ficheros que crean los módulos están agrupado dentro de localizaciones específicicas bajo la estructura de directorios de Drupal. Esto es que, en el sistema de ficheros de la instalación de Drupal, los módulos Drupal deben residir en unos pocos lugares.

Cuando Drupal necesita información acerca de sus módulos, mirará en estos lugares predeterminados. Cada módulo está contenido en su propio directorio, y tiene al menos 2 ficheros—uno describiendo el contenido del módulo y uno ó más ficheros conteniendo código y otro material de soporte.

Antes de que un módulo pueda ser usado, debe ser activado por un administrador de Drupal. Sin embargo, una vez un módulo es activado, entonces es cargado cuando se le requiere y Drupal tramita las solicitudes al módulo.

Módulos del Núcleo

Algunos módulo son tan importantes que eliminándolos desactivarías funcionalidades esenciales para el funcionamiento de Drupal. También, hay módulos que proporcionan funcionalidades necesarias por una amplia variedad de sistemas. Estos dos grupos de módulos, son mantenidos por el equipo de desarrolladores de Drupal, y son colectivamente referidos como los Módulos del Núcleo de Drupal. Estos módulos están incluidos por defecto en la instalación de Drupal, y tienen un activo mantenimiento y desarrollo por la comunidad Drupal.

A pesar de este importante rol, hay pocas distinciones entre los Módulos del Núcleo de Drupal y cualquier otro módulo. Ellos siguen las mismas directrices y usan la misma API.

Desde la sección de administración de Drupal, es posible ver la lista de los módulos del núcleo en Administración → Construcción del Sitio → Módulos.

Una de las maravillas de la arquitectura de Drupal es la facilidad con que varios módulos interactúan. Usando la arquitectura de hook, los servicios que los módulos proporcionan pueden ser tejidos juntos para crear robustas funcionalidades sin copiosas cantidades de código.

Los módulos del core proporcionan una excelente referencia para saber cómo el código de Drupal debe ser escrito.

Hooks

¿Cómo sabe Drupal cuando invocar a un módulo para manejar una solicitud particular?. Esto es hecho a través del mecanismo de hooks, que se va a examinar cuidadosamente en este manual. Para empezar, una breve explicación de cómo funcionan los hooks. Cuando Drupal gestiona una solicitud desde un usuario, procede a través de una serie de pasos. Por ejemplo, el núcleo de Drupal primero inicia la aplicación, definiendo variables críticas y funciones frecuentemente usadas. Lo siguiente es, cargar librerías críticas, temas y módulos. Lo siguiente es, continuar procesando la solicitud, mapeando la URI solicitada al código que la maneja y así. Después, aplica un tema a los datos, formateando información como salida. Finalmente, devuelve la salida al navegador del usuario.

En este paso a paso, hay momentos predefinidos en los que Drupal ejecuta hooks. ¿Qué significa esto?. En resumen, significa que Drupal examina algunos ó todos los módulos actualmente activados que siguen específicos y predefinidos patrones. Algunos tienen enlazado a este proceso un método “callback”.

Por ejemplo, mientras que se está creando el contenido para una página vista, Drupal podría escanear módulos para funciones llamadas <modulename>_block() y <modulename>_view() (donde <modulename> es reemplazado por el nombre de cada módulo que chequea)

Los módulos que contienen tales funciones son aquellos que implementan hook_block() y hook_view().

Cuando Drupal encuentra tales funciones, las ejecuta, y usa los datos de estas funciones para devolver una respuesta que enviar al usuario. Drupal continúa su procesamiento de la solicitud paso por paso, quizás ejecutando muchos otros hooks. Una vez que todos los pasos han sido completados y una respuesta ha sido enviada al usuario, Drupal se limpia a sí mismo y termina la ejecución.

Los módulos pueden definir sus propios hooks, que otros módulos pueden usar. De este modo, el mecanismo de hook puede ser extendido para proporcionar un personalizado

comportamiento complejo.

Cuando un módulo proporciona una función que empareja una firma hook, se dice que este módulo implementa este hook. Por ejemplo, imagina que Drupal tiene un hook llamado hook_example. Si se definiéra un módulo llamado mimodulo que contuviera una función llamada mimodulo_example(), se estaría implementando hook_example().

2. Creación de Bloques

Esta sección tiene un doble objetivo: introducir a la API de bloques y crear un primer módulo. Y en el siguiente capítulo se verá cómo instalar dicho módulo.

En Drupal, cada módulo está contenido en su propio directorio. Esto simplifica la organización; todos los ficheros de un módulo están localizados en un lugar. Así que se va a crear un directorio que sea descriptivo del bloque que se va a crear, en este caso mibloque

```
cd sites/all/modules
mkdir mibloque
```

Una vez se ha creado el directorio, es posible empezar a crear ficheros para el módulo. El primer fichero a crear será el fichero .info.

2.1 Creando un fichero .info

El fichero .info es escrito como un fichero PHP INI, que es un formato para configuraciones simples.

El fichero .info debe seguir las convenciones de nombres estándar para módulos. Debe ser nombrado <modulename>.info, donde <modulename> es el mismo que el nombre del directorio. El fichero, por tanto, será llamado mibloque.info.

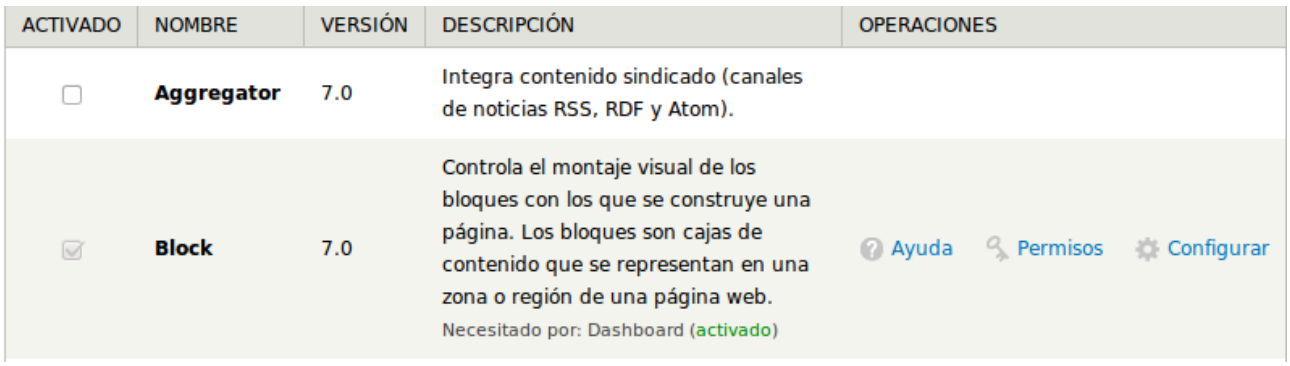
Lo siguiente son los contenidos de mibloque.info:

```
;$Id$
name = "Mi Bloque"
description = "Muestra un bloque creado desde la API"
core = 6.x
```

Este fichero no es particularmente grande. La primera línea del fichero es, a primera vista, la más críptica. Sin emabargo, su función es mundana: es la variable para servidor CVS o Subversion de Drupal.

Las siguientes 3 directivas proporcionan información del módulo a Drupal

La directiva name proporciona un nombre “legible por humanos” para el módulo. Anteriormente, ya se ha visto brevemente cómo se activa un módulo. Los nombres de los módulos que se vieron se extrajeron de la directiva name de los ficheros .info. Aquí hay un ejemplo:



En el pantallazo los nombres “Aggregator” y “Block” son tomado de las directivas name de sus respectivos fichero .info.

Otra directiva que también aparece en el pantallazo es description (“Descripción”).

La tercera directiva es core. Esta directiva especifica qué versión de Drupal es requerida para que el módulo funcione de manera apropiada. El valor 6.x indica que el módulo funcionará en Drupal 6 (incluidas sus revisiones). En muchos casos, el empaquetador de Drupal será capaz de establecer esto correctamente de manera automática. Pero los desarrolladores de Drupal sugieren que esta directiva sea asignada manualmente para quienes trabajan desde CVS.

2.2 Un fichero .module

Como se mencionó en el primer capítulo, hay dos ficheros que cualquier módulo debe tener (aunque muchos módulos tienen más). El primero, es el fichero .info, del que ya se ha hablado. El segundo fichero es el fichero .module, que es un fichero script en PHP. Este fichero implementa un puñado de funciones hook que Drupal llamará en predeterminadas veces durante una solicitud.

Aquí, se creará un fichero .module que mostrará uns pequeña y formateada sección de información. Después, en este capítulo, se configurará Drupal para mostrar esto a los visitantes del sitio.

2.3 El objetivo: Un block hook

Para este muy primer módulo, se implementará la función hook_block(). En dialecto Drupal, un bloque es un trozo de información auxiliar que es mostrada en una página a lo largo de la página principal de contenido. ¿Suena confuso? Un ejemplo podría ayudar. Piense en su sitio web favorito de noticias. En una típica página de artículo, el texto del artículo es mostrado en la mitad de la página. Pero en los laterales y quizás arriba y abajo, hay otros trozos de información: un menú del sitio, una lista de enlaces a artículos relacionados, enlaces a comentarios ó foros acerca del artículo, etc. En Drupal, estas piezas extra son tratadas como bloques.

2.4 Empezando el .module

Como ya se mencionó, Drupal sigue una codificación rigurosa y estándares de documentación (http://drupal.org/coding-standards). En este manual, se hará todo lo posible por seguir estos estándares. Así que al empezar el módulo, la primera cosa que se hace es proporcionar alguna documentación para el API.

Se empieza con el fichero mibloque.module:

```
<?php
// $Id$
/**
* @file
* Module for show data in a block
* This module provides block content builded from the API
*/
```

Después del PHP tag “<?php” se encuentra la palabra clave para el control de versiones:

// Id

Cuando el módulo sea chequeado dentro del CVS de Drupal, la información acerca de la actual revisión es fijada ahí.

La tercera parte de este ejemplo es la documentación del API. Esta documentación es contenido en un bloque de comentario especial, que comienza con /** y finaliza con */. Cualquier cosa entre esto es tratada como documentación. Programas de extracción como Doxygen pueden extraer esta información y crear información de programación útil para el usuario final.

2.5 La implementación de hook_block()

El módulo muestra información dentro de un bloque Drupal. Para hacer esto, se necesita implementar las función hook_block().

Todos los métodos hook siguen la convención de nombre: <module name>_<hook name>. Así, este hook de bloque se llamará mibloque_block().

```
function mibloque_block($op='list', $delta=0, $edit=array()) {
  switch ($op) {
  case 'list':
    $blocks[0]['info'] = t('My Block');
    return $blocks;
  case 'view':
    $blocks['subject'] = t('Título de Mi Bloque');
    $blocks['content'] = t('Cuerpo de Mi Bloque');
    return $blocks;
  }
}
```

La variable $op puede tener varios valores, en este caso, list y view. Si $op='list' entonces se está listando el bloque desde la administración de bloques (Administración → Configuración del Sitio → Bloques), en ese caso se visualiza como t('My Block'). Si $op='view' entonces se está visualizando el bloque y para este caso se define el valor de las variables asunto ($blocks['subject']) y contenido ($blocks['content']).

En este ejemplo solo se va a dar un valor a la cadena con la que se identificará este bloque en la administración de bloques.

3. Instalación de Módulos

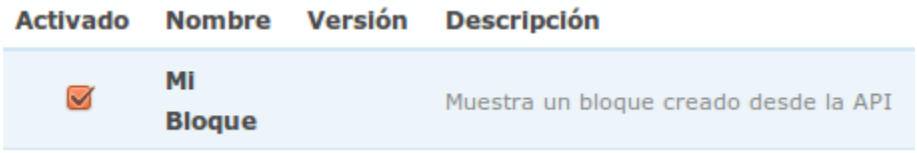
Para poder visualizar el resultado del módulo escrito es necesario aprender a instalar módulos.

El directorio de éste módulo debe estar en el directorio drupal/sites/all/modules, nótese que también existe drupal/modules pero en ese directorio solo deben estar los módulos

del núcleo de Drupal. En caso de que no esté se copia:

```
cp -r mibloque /var/www/drupal/sites/all/modules
```

Ahora ya se puede ir a Administración → Construcción del Sitio → Módulos y buscar el módulo. Se activa y se pulsa en guardar configuración.



Lo siguiente es activar el bloque desde Administración → Estructura → Bloques y ya se puede ver el resultado:

4. Form API

Ahora un tutorial paso a paso para aprender la API de formularios. Lo primero es descargar e instalar el módulo examples de http://drupal.org/project/examples. Una vez instalado ya es posible navegar por los ejemplos desde http://localhost/drupal6/examples/form_example/tutorial. Empiécese por el primero:

4.1 Formulario Básico

Este es un formulario muy básico que será extendido en los siguientes ejemplos. Ahora mostraré el código relacionado con este ejemplo:

```
...
  $items['examples/form_example/tutorial/1'] = array(
    'title' => t('#1'),
    'page callback' => 'drupal_get_form',
    'page arguments' => array('form_example_tutorial_1'),
    'access callback' => TRUE,
    'description' => t('Tutorial 1: Simplest form'),
    'type' => MENU_DEFAULT_LOCAL_TASK,
    'file' => 'form_example_tutorial.inc',
  );
…
```

La primera función implementa hook_menu y, dicho de manera muy somera, proporciona el acceso url desde el que se visualizará la página que implementa el formulario, en este caso: examples/form_example/tutorial/1. Al llamar a esta página se llama a la función drupal_get_form con array('form_example_tutorial_1') como argumento, esta función está almacenada en el fichero form_example_tutorial.inc que es una librería de funciones php.

Ahora el contenido de form_example_tutorial_1

```
…
function form_example_tutorial_1($form_state) {

  $form['description'] = array(
```

```
    '#type' => 'item',
    '#title' => t('A form with nothing but a textfield'),
  );
  // This is the first form element. It's a textfield with a label, "Name"
  $form['name'] = array(
    '#type' => 'textfield',
    '#title' => t('Name'),
  );
  return $form;
}
...
```

form_example_tutorial_1 es un array con 2 claves: description y name, description tiene es un texto que introduce al formulario y name es el campo de texto del formulario en sí.

4.2 Formulario Básico con Botón Submit

Ahora se añade un botón submit al formulario anterior.

```
function form_example_tutorial_2(&$form_state) {
  $form['description'] = array(
    '#type' => 'item',
    '#title' => t('A simple form with a submit button'),
  );

  $form['name'] = array(
    '#type' => 'textfield',
    '#title' => t('Name'),
  );

  // Añade un botón simple de envío que refresca el formulario y resetea los contenidos – este es el comportamiento por defecto para formularios.

  $form['submit'] = array(
    '#type' => 'submit',
    '#value' => 'Submit',
  );
  return $form;
}
```

En este ejemplo tan solo se ha añadido el botón submit.

4.3 Un formulario básico con fieldset

Ahora se verá cómo añadir fieldset (conjunto de campos).

```
/**
 * Se establece un elemento fieldset y 2 elementos campo de texto dentro
 * de él, uno para el nombre y otro para el apellido. Esto ayuda a agrupar
 * contenido relacionado.
 *
 * Estudiando el código de debaso se verá que se renombra el array de los
 * campos first name y last name bajo el array $form['name'].
 */
function form_example_tutorial_3(&$form_state) {
  $form['description'] = array(
```

```
    '#type' => 'item',
    '#title' => t('A form with a fieldset'),
  );

  $form['name'] = array(
    '#type' => 'fieldset',
    '#title' => t('Name'),
  );
  $form['name']['first'] = array(
    '#type' => 'textfield',
    '#title' => t('First name'),
  );
  $form['name']['last'] = array(
    '#type' => 'textfield',
    '#title' => t('Last name'),
  );


  $form['submit'] = array(
    '#type' => 'submit',
    '#value' => 'Submit',
  );
  return $form;
}
```

Con un fieldset se agrupa contenido y en código esto se implementa anidando arrays.

4.4 Formulario Básico con Campos Obligatorios

```
function form_example_tutorial_4(&$form_state) {
  $form['description'] = array(
    '#type' => 'item',
    '#title' => t('A form with validation'),
  );

  $form['name'] = array(
    '#type' => 'fieldset',
    '#title' => t('Name'),
    // Make the fieldset collapsible.
    '#collapsible' => TRUE,
    '#collapsed' => FALSE,
  );

  // Make these fields required.
  $form['name']['first'] = array(
    '#type' => 'textfield',
    '#title' => t('First name'),
    '#required' => TRUE,
  );
  $form['name']['last'] = array(
    '#type' => 'textfield',
    '#title' => t('Last name'),
    '#required' => TRUE,
  );

  $form['submit'] = array(
    '#type' => 'submit',
    '#value' => 'Submit',
  );
  return $form;
```

```
}
```

La obligatoriedad se implementa con una clave en el array llamada #required.

4.5 Formulario Básico con Atributos de Elementos Adicionales

```
/*
 * Se muestra como añadir atributos adicionales en elmentos de texto
 *
 * Para un ejemplo más extenso en tipos de elementos
 * ver http://drupal.org/node/751826
 *
 * Ver la referencia completa de formulario
 * en http://api.drupal.org/api/file/developer/topics/forms_api.html

 */
function form_example_tutorial_5(&$form_state) {
  $form['description'] = array(
    '#type' => 'item',
    '#title' => t('A form with additional attributes'),
    '#description' => t('This one adds #default_value and #description'),
  );
  $form['name'] = array(
    '#type' => 'fieldset',
    '#title' => t('Name'),
    '#collapsible' => TRUE,
    '#collapsed' => FALSE,
  );

  $form['name']['first'] = array(
    '#type' => 'textfield',
    '#title' => t('First name'),
    '#required' => TRUE,
    '#default_value' => "First name", // added default value.
    '#description' => "Please enter your first name.", // added description
    '#size' => 20, // added
    '#maxlength' => 20, // added
  );
  $form['name']['last'] = array(
    '#type' => 'textfield',
    '#title' => t('Last name'),
    '#required' => TRUE,
  );
  $form['submit'] = array(
    '#type' => 'submit',
    '#value' => 'Submit',
  );
  return $form;
}
```

Destacar '#default_value' => "First name", que es donde se asigna que el valor por defecto es “First name”. De nuevo una nueva clave en el array.

4.6 Formulario Básico con Manejador de la Validación

Ahora se va a introducir un campo fecha en el que solo se puedan introducir valores entre 1900 y el 2000.

```
function form_example_tutorial_6(&$form_state) {
  $form['description'] = array(
    '#type' => 'item',
    '#title' => t('A form with a validation handler'),
  );

  $form['name'] = array(
    '#type' => 'fieldset',
    '#title' => t('Name'),
    '#collapsible' => TRUE,
    '#collapsed' => FALSE,
  );
  $form['name']['first'] = array(
    '#type' => 'textfield',
    '#title' => t('First name'),
    '#required' => TRUE,
    '#default_value' => "First name",
    '#description' => "Please enter your first name.",
    '#size' => 20,
    '#maxlength' => 20,
  );
  $form['name']['last'] = array(
    '#type' => 'textfield',
    '#title' => t('Last name'),
    '#required' => TRUE,
  );

  // New form field added to permit entry of year of birth.
  // The data entered into this field will be validated with
  // the default validation function.
  $form['year_of_birth'] = array(
    '#type' => 'textfield',
    '#title' => "Year of birth",
    '#description' => 'Format is "YYYY"',
  );

  $form['submit'] = array(
    '#type' => 'submit',
    '#value' => 'Submit',
  );
  return $form;
}



/**
 * Ahora se añade una función para validar los datos introducidos dentro del
 * campo "year of birth" para asegugarse que está entre los valores 1900 y
 * 2000. Si no es así, se muestra un error. El valor es $form_state['values']
 * (ver http://drupal.org/node/144132#form-state).
 *
 * Fíjese en el nombre de la función. Es simplemente el nombre del formulario
 * seguido por '_validate'. Este es siempre el nombre por defecto de la función
 * validación. Una lista alternativa de funciones de validación podría haber
 * sido proporcionada por $form['#validate'].
 */
function form_example_tutorial_6_validate($form, &$form_state) {
  $year_of_birth = $form_state['values']['year_of_birth'];
  if ($year_of_birth && ($year_of_birth < 1900 || $year_of_birth > 2000)) {
    form_set_error('year_of_birth', 'Enter a year between 1900 and 2000.');
```

```
  }
}
```

Ahora aparece al final del código una nueva función (form_example_tutorial_6_validate) que es la que hace la validación.

4.7 Formulario con un manejador de submit

```
function form_example_tutorial_7($form_state) {
  $form['description'] = array(
    '#type' => 'item',
    '#title' => t('A form with a submit handler'),
  );
  $form['name'] = array(
    '#type' => 'fieldset',
    '#title' => t('Name'),
    '#collapsible' => TRUE,
    '#collapsed' => FALSE,
  );
  $form['name']['first'] = array(
    '#type' => 'textfield',
    '#title' => t('First name'),
    '#required' => TRUE,
    '#default_value' => "First name",
    '#description' => "Please enter your first name.",
    '#size' => 20,
    '#maxlength' => 20,
  );
  $form['name']['last'] = array(
    '#type' => 'textfield',
    '#title' => t('Last name'),
    '#required' => TRUE,
  );
  $form['year_of_birth'] = array(
    '#type' => 'textfield',
    '#title' => "Year of birth",
    '#description' => 'Format is "YYYY"',
  );
  $form['submit'] = array(
    '#type' => 'submit',
    '#value' => 'Submit',
  );
  return $form;
}


/**
 * Función Validación.
 */
function form_example_tutorial_7_validate($form, &$form_state) {
    $year_of_birth = $form_state['values']['year_of_birth'];
    if ($year_of_birth && ($year_of_birth < 1900 || $year_of_birth > 2000)) {
        form_set_error('year_of_birth', 'Enter a year between 1900 and 2000.');
    }
}

/**
 * Añade una función submit al formulario para enviar un mensaje de
 * 'completado con éxito' a la pantalla.
```

```
 */

function form_example_tutorial_7_submit($form, &$form_state) {
   drupal_set_message(t('The form has been submitted. name="@first @last", year of
birth=@year_of_birth',
     array('@first' => $form_state['values']['first'], '@last' => $form_state['values']['last'], '@year_of_birth' =>
$form_state['values']['year_of_birth'])));
}
```

En este ejemplo, de nuevo aparece validate, aunque lo importante es form_example_tutorial_7_submit. Esta función imprime un mensaje con los valores introducidos en el formulario. Este manejador también puede servir para grabar los datos como se verá en el último ejemplo de este tutorial.

4.8 Un Formulario con un Botón de Reseteo

```
function form_example_tutorial_8(&$form_state) {
  $form['description'] = array(
    '#type' => 'item',
    '#title' => t('A form with a "reset" button'),
  );

  $form['name'] = array(
    '#type' => 'fieldset',
    '#title' => t('Name'),
    '#collapsible' => TRUE,
    '#collapsed' => FALSE,
  );

  // Elimina la propiedad #required y en vez de ésta usa la función
  // validación
  $form['name']['first'] = array(
    '#type' => 'textfield',
    '#title' => t('First name'),
    '#default_value' => "First name",
    '#description' => "Please enter your first name.",
    '#size' => 20,
    '#maxlength' => 20,
  );

  // Elimina la propiedad #required y en vez de ésta usa la función
  // validación
  $form['name']['last'] = array(
    '#type' => 'textfield',
    '#title' => t('Last name'),
  );
  $form['year_of_birth'] = array(
    '#type' => 'textfield',
    '#title' => "Year of birth",
    '#description' => 'Format is "YYYY"',
  );
  $form['submit'] = array(
    '#type' => 'submit',
    '#value' => 'Submit',
  );
  // Añade un nuevo botón para limpiar el formulario. La propiedad #validate
  // direcciona el formulario para usar una función de validación específica
  // para este botón en vez de usar el manejador por defecto.
  $form['clear'] = array(
    '#type' => 'submit',
    '#value' => 'Reset form',
```

```
    '#validate' => array('form_example_tutorial_8_clear'),
  );

  return $form;
}

/**
 * Función de Validación para el botón de reset.
 *
 * Estableciendo el valor de $form_state['rebuild'] a TRUE dice que se está
 * yendo a reconstruir el formulario. También se limpian los valores
 * existentes.
 */
function form_example_tutorial_8_clear($form, &$form_state) {
    $form_state['rebuild'] = TRUE;
    unset($form_state['values']);
}

/**
 * Hace que en los campos first name y last name usados por la función
 * validación, haya seguridad de que sus valores hayan sido introducidos.
 * Esto causa que el nombre de los campos se muestre en rojo si quedaron en
 * blanco.
 *
 * De forma más extensa, una función validación puede chequear
 * interdependencias entre campos. Por ejemplo, podría estar OK introducir
 * solo un last name, o si se introduce un last name que sea necesario
 * introducir un first name.
 */
function form_example_tutorial_8_validate($form, &$form_state) {
    $year_of_birth = $form_state['values']['year_of_birth'];
    $first_name = $form_state['values']['first'];
    $last_name = $form_state['values']['last'];
    if (!$first_name) {
        form_set_error('first', 'Please enter your first name.');
    }
    if (!$last_name) {
        form_set_error('last', 'Please enter your last name.');
    }
    if ($year_of_birth && ($year_of_birth < 1900 || $year_of_birth > 2000)) {
        form_set_error('year_of_birth', 'Enter a year between 1900 and 2000.');
    }
}

function form_example_tutorial_8_submit($form, &$form_state) {
  drupal_set_message(t('The form has been submitted. name="@first @last", year of birth=@year_of_birth',
    array('@first' => $form_state['values']['first'], '@last' => $form_state['values']['last'], '@year_of_birth' =>
$form_state['values']['year_of_birth'])));
}
```

Este código introduce algunas novedades. Anteriormente los campos obligatorios eran añadidos mediante '#required' => TRUE, sin embargo, aquí se contempla el siguiente código en form_example_tutorial_8_validate:

```
    if (!$last_name) {
        form_set_error('last', 'Please enter your last name.');
    }
```

No obstante, la funcionalidad del reseteo se implementa mediante $form['clear'] y la llamada mediante validate a form_example_tutorial_8_clear.

4.9 Un Formulario con Campos Dinámicos

En este formulario los fieldset de tipo nombre van a ser dinámicos, esto es, se irán añadiendo fieldset según se dé al botón añadir nombre. Ahora el código para hacer esto:

```
function form_example_tutorial_9(&$form_state) {
  $form['description'] = array(
    '#type' => 'item',
    '#title' => t('A form with dynamically added new fields'),
  );

  $form['name'] = array(
    '#type' => 'fieldset',
    '#title' => t('Name'),
    '#collapsible' => TRUE,
    '#collapsed' => FALSE,
  );

  $form['name']['first'] = array(
    '#type' => 'textfield',
    '#title' => t('First name'),
    '#default_value' => !empty($form_state['values']['first']) ? $form_state['values']['first'] : '', // changed
    '#description' => "Please enter your first name.",
    '#size' => 20,
    '#maxlength' => 20,
  );
  $form['name']['last'] = array(
    '#type' => 'textfield',
    '#title' => t('Last name'),
    '#default_value' => !empty($form_state['values']['last']) ? $form_state['values']['last'] : '', // added
  );
  $form['year_of_birth'] = array(
    '#type' => 'textfield',
    '#title' => "Year of birth",
    '#description' => 'Format is "YYYY"',
    '#default_value' => !empty($form_state['values']['year_of_birth']) ? $form_state['values']['year_of_birth'] :
'', // added
  );

  // Añade nuevos elementos al formulario si $form_state['storage']['new_name']
  // es establecido.
  // Esto ocurre cuando el botón "add new name" se clica.
  if (isset($form_state['storage']['new_name'])) {
    $form['name2'] = array(
      '#type' => 'fieldset',
      '#title' => t('Name #2'),
      '#collapsible' => TRUE,
      '#collapsed' => FALSE,
    );
    $form['name2']['first2'] = array(
      '#type' => 'textfield',
      '#title' => t('First name'),
      '#description' => "Please enter your first name.",
      '#size' => 20,
      '#maxlength' => 20,
      '#default_value' => !empty($form_state['values']['first2']) ? $form_state['values']['first2'] : '',
    );
    $form['name2']['last2'] = array(
      '#type' => 'textfield',
      '#title' => t('Last name'),
      '#default_value' => !empty($form_state['values']['last2']) ? $form_state['values']['last2'] : '',
    );
```

```
    $form['year_of_birth2'] = array(
      '#type' => 'textfield',
      '#title' => "Year of birth",
      '#description' => 'Format is "YYYY"',
      '#default_value' => !empty($form_state['values']['year_of_birth2']) ? $form_state['values']
['year_of_birth2'] : '',
    );
  }

  $form['submit'] = array(
    '#type' => 'submit',
    '#value' => 'Submit',
  );
  $form['clear'] = array(
    '#type' => 'submit',
    '#value' => 'Reset form',
    '#validate' => array('form_example_tutorial_9_clear'),
  );

  // Adds "Add another name" button, if it hasn't already been clicked.
  if (empty($form_state['storage']['new_name'])) {
    $form['new_name'] = array(
      '#type' => 'submit',
      '#value' => 'Add another name',
      '#validate' => array('form_example_tutorial_9_new_name'),
    );
  }

  return $form;
}

/**
 * Creando un nuevo elemento 'new_name' en el array $form_state['storage']
 * establece el valor usado para determinar si mostrar los nuevos campos en
 * el formulario y ocultar el botón "Add another name".
 *
 * $form_state['storage'] es solo un lugar donde almacenar información de
 * repuesto. Cualquier clave puede ser creada en este array.
 */
function form_example_tutorial_9_new_name($form, &$form_state) {
    $form_state['storage']['new_name'] = TRUE;
    $form_state['rebuild'] = TRUE;
}

function form_example_tutorial_9_clear($form, &$form_state) {
  // Asegura que los campos quedan en blanco después de pulsar el botón de reset
  unset($form_state['values']);
  // Asegura que el botón reset elimina el botón new_name.
  unset($form_state['storage']);
  // Reconstruye el formulario
  $form_state['rebuild'] = TRUE;
}

/**
 * Añade lógica para validar el formulario para chequear la validez de los
 * nuevos campos si estos existen.
 */
function form_example_tutorial_9_validate($form, &$form_state) {
  $year_of_birth = $form_state['values']['year_of_birth'];
  $first_name = $form_state['values']['first'];
  $last_name = $form_state['values']['last'];
  if (!$first_name) {
```

```
    form_set_error('first', 'Please enter your first name.');
  }
  if (!$last_name) {
    form_set_error('last', 'Please enter your last name.');
  }
  if ($year_of_birth && ($year_of_birth < 1900 || $year_of_birth > 2000)) {
    form_set_error('year_of_birth', 'Enter a year between 1900 and 2000.');
  }
  if ($form_state['storage']['new_name']) {
    $year_of_birth = $form_state['values']['year_of_birth2'];
    $first_name = $form_state['values']['first2'];
    $last_name = $form_state['values']['last2'];
    if (!$first_name) {
      form_set_error('first2', 'Please enter your first name.');
    }
    if (!$last_name) {
      form_set_error('last2', 'Please enter your last name.');
    }
    if ($year_of_birth && ($year_of_birth < 1900 || $year_of_birth > 2000)) {
      form_set_error('year_of_birth2', 'Enter a year between 1900 and 2000.');
    }
  }
}
/**
 * Función Submit.
 *
 * Comentando unset($form_state['storage'] y, añadiendo un nuevo conjunto
 * de campos name y enviándo el formulario, causaría que el formulario se
 * reconstruiría con los valores existente, porque en Drupal 6 si
 * $form_state['storage'] está asignado, $form_state['rebuild'] está también
 * automaticamente asignado, causando que los campos del formularios se
 * obtienen de los valores encontrados en $form_state['values'].
 */
function form_example_tutorial_9_submit($form, &$form_state) {
  unset($form_state['storage']);
  drupal_set_message(t('The form has been submitted. name="@first @last", year of birth=@year_of_birth',
    array('@first' => $form_state['values']['first'], '@last' => $form_state['values']['last'], '@year_of_birth' =>
$form_state['values']['year_of_birth'])));
  if (!empty($form_state['values']['first2'])) {
      drupal_set_message(t('Second name: name="@first @last", year of birth=@year_of_birth',
    array('@first' => $form_state['values']['first2'], '@last' => $form_state['values']['last2'], '@year_of_birth' =>
$form_state['values']['year_of_birth2'])));
  }
}
```

El formulario tiene un fieldset que se añade y se elimina dinámicamente. Para añadirlo hay que pulsar al botón “Add another name”. Este botón está definido como un campo tipo submit que valida a la función form_example_tutorial_9_new_name. Esta función establece $form_state['storage']['new_name'] a TRUE y reconstruye el formulario. Al reconstruirse el formulario muchos elementos del mismo que habían sido escapados por sentencias if se van a incluir (el nuevo fieldset) y desparecerá el submit “Add another name”. Para eliminar el nuevo fieldset se pulsa en “Reset form” que es un botón de tipo submit llama a la función form_example_tutorial_9_clear que resetea por completo el formulario.

form_example_tutorial_9_validate valida la obligatoriedad y corrección de campos como en anteriores ejercicios. Y form_example_tutorial_9_submit imprime un mensaje en pantalla con los valores introducidos.

4.10 Un Formulario de Múltiples Pasos

Ahora toca aprender a crear formualarios de múltiples pasos, esto es, que se debe pasar por más de una pantalla para completar el formulario.

```
/**
 *
 * Más sobre formularios de múltiples pasos
 * en http://pingvision.com/blog/ben-jeavons/2009/multi-step-forms-drupal-6-using-variable-functions
 *
 */

/**
 * Añade la lógica al constructor del formulario para crearlo en dos páginas
 * Chequea un valor en $form_state['storage'] para determinar si muestra la
 * página 2.
 */

function form_example_tutorial_10($form_state) {

  // Se muestra la página 2 si $form_state['storage']['page_two'] está asignado
  if (isset($form_state['storage']['page_two'])) {
    return form_example_tutorial_10_page_two();
  }
  $form['description'] = array(
    '#type' => 'item',
    '#title' => t('A basic multistep form (page 1)'),
  );

  // La página 1 es mostrada si $form_state['storage']['page_two'] no está
  // asignado
  $form['name'] = array(
    '#type' => 'fieldset',
    '#title' => t('Name'),
    '#collapsible' => TRUE,
    '#collapsed' => FALSE,
  );
  $form['name']['first'] = array(
    '#type' => 'textfield',
    '#title' => t('First name'),
    '#default_value' => !empty($form_state['values']['first']) ? $form_state['values']['first'] : '',
    '#description' => "Please enter your first name.",
    '#size' => 20,
    '#maxlength' => 20,
  );
  $form['name']['last'] = array(
    '#type' => 'textfield',
    '#title' => t('Last name'),
    '#default_value' => !empty($form_state['values']['last']) ? $form_state['values']['last'] : '',
  );
  $form['year_of_birth'] = array(
    '#type' => 'textfield',
    '#title' => "Year of birth",
    '#description' => 'Format is "YYYY"',
    '#default_value' => !empty($form_state['values']['year_of_birth']) ? $form_state['values']['year_of_birth'] :
'',
  );
  // Añade nuevos elementos al formulario
  if (!empty($form_state['storage']['new_name'])) {
    $form['name2'] = array(
      '#type' => 'fieldset',
      '#title' => t('Name #2'),
      '#collapsible' => TRUE,
```

```
      '#collapsed' => FALSE,
    );
    $form['name2']['first2'] = array(
      '#type' => 'textfield',
      '#title' => t('First name'),
      '#description' => "Please enter your first name.",
      '#size' => 20,
      '#maxlength' => 20,
      '#default_value' => !empty($form_state['values']['first2']) ? $form_state['values']['first2'] : '',
    );
    $form['name2']['last2'] = array(
      '#type' => 'textfield',
      '#title' => t('Last name'),
      '#default_value' => !empty($form_state['values']['last2']) ? $form_state['values']['last2'] : '',
    );
    $form['year_of_birth2'] = array(
      '#type' => 'textfield',
      '#title' => "Year of birth",
      '#description' => 'Format is "YYYY"',
      '#default_value' => !empty($form_state['values']['year_of_birth2']) ? $form_state['values']
['year_of_birth2'] : '',
    );
  }
  $form['clear'] = array(
    '#type' => 'submit',
    '#value' => 'Reset form',
    '#validate' => array('form_example_tutorial_10_clear'),
  );
  if (empty($form_state['storage']['new_name'])) {
    $form['new_name'] = array(
      '#type' => 'submit',
      '#value' => 'Add another name',
      '#validate' => array('form_example_tutorial_10_new_name'),
    );
  }
  $form['next'] = array(
    '#type' => 'submit',
    '#value' => 'Next >>',
  );
  return $form;
}

// Nueva función creada para hacer el código más manejable
function form_example_tutorial_10_page_two() {
  $form['description'] = array(
    '#type' => 'item',
    '#title' => t('A basic multistep form (page 2)'),
  );

  $form['color'] = array(
    '#type' => 'textfield',
    '#title' => 'Your favorite color',
  );
  $form['submit'] = array(
    '#type' => 'submit',
    '#value' => 'Submit',
  );
  return $form;
}

function form_example_tutorial_10_new_name($form, &$form_state) {
  $form_state['storage']['new_name'] = TRUE;
```

```
  $form_state['rebuild'] = TRUE; // se escapará a la función submit por defecto
}

function form_example_tutorial_10_clear($form, &$form_state) {
  unset($form_state['values']);
  unset($form_state['storage']);

  $form_state['rebuild'] = TRUE;
}

/**
 * The validate function now validates page 2 as well.
 */
function form_example_tutorial_10_validate($form, &$form_state) {
  // Validate page 2 here
  if (isset($form_state['storage']['page_two'])) {
    $color = $form_state['values']['color'];
    if (!$color) {
      form_set_error('color', 'Please enter a color.');
    }
    return;
  }

  $year_of_birth = $form_state['values']['year_of_birth'];
  $first_name = $form_state['values']['first'];
  $last_name = $form_state['values']['last'];
  if (!$first_name) {
    form_set_error('first', 'Please enter your first name.');
  }
  if (!$last_name) {
    form_set_error('last', 'Please enter your last name.');
  }
  if ($year_of_birth && ($year_of_birth < 1900 || $year_of_birth > 2000)) {
    form_set_error('year_of_birth', 'Enter a year between 1900 and 2000.');
  }
  if ($form_state['storage']['new_name']) {
    $year_of_birth = $form_state['values']['year_of_birth2'];
    $first_name = $form_state['values']['first2'];
    $last_name = $form_state['values']['last2'];
    if (!$first_name) {
      form_set_error('first2', 'Please enter your first name.');
    }
    if (!$last_name) {
      form_set_error('last2', 'Please enter your last name.');
    }
    if ($year_of_birth && ($year_of_birth < 1900 || $year_of_birth > 2000)) {
      form_set_error('year_of_birth2', 'Enter a year between 1900 and 2000.');
    }
  }
}

/**
 * Modifica esta función de tal manera que responderá apropiadamente basado
 * en que página era enviada. Si la primera página está siendo enviada,
 * los valores son guardados en el array 'storage' y el formulario es
 * automáticamente recargado.
 *
 * Si la página 2 fuera enviada, se muestra un mensaje y redirecciona el
 * usuario a otra página.
 */
function form_example_tutorial_10_submit($form, &$form_state) {
  // Handle page 1 submissions
```

```
  if ($form_state['clicked_button']['#id'] == 'edit-next') {
    $form_state['storage']['page_two'] = TRUE; // Se establece esto para
                                               // determinar qué elementos
                                               // mostrar cuando la página se
                                               // recarga.

    // Valores por debajo del array $form_state['storage'] son guardados
    // para traerlos en las subsiguientes páginas en el formulario.
    $form_state['storage']['page_one_values'] = $form_state['values'];
  }
  // Maneja los envíos a la página 2.
  else {
    /*
     Normalmente, algún código iría aquí para alterar la base de datos con
     los datos recogidos en el formulario. Se establece un mensaje con
     drupal_set_message() para validar el código en funcionamiento.
     */
    $page_one_values = $form_state['storage']['page_one_values'];
    drupal_set_message(t('The form has been submitted. name="@first @last", year of
birth=@year_of_birth',
      array('@first' => $page_one_values['first'], '@last' => $page_one_values['last'], '@year_of_birth' =>
$page_one_values['year_of_birth'])));

    if (!empty($page_one_values['first'])) {
      $first2 = isset($page_one_values['first2']) ? $page_one_values['first2'] : '';
      $last2  = isset($page_one_values['last2']) ? $page_one_values['last2'] : '';
      $year2  = isset($page_one_values['year_of_birth2']) ? $page_one_values['year_of_birth2'] : '';
      drupal_set_message(t('Second name: name="@first @last", year of birth=@year_of_birth',
        array('@first' => $first2, '@last' => $last2, '@year_of_birth' => $year2)));
    }
    drupal_set_message(t('And the favorite color is @color', array('@color' => $form_state['values']['color'])));

    // $form_state['storage'] must be unset for redirection to occur. Otherwise
    // $form_state['rebuild'] is automatically set and this form will be
    // rebuilt.
    unset($form_state['storage']);
    $form_state['redirect'] = 'node'; // Redirects the user to /node.
  }
}
```

Este código lleva implementada la funcionalidad de múltiples pasos, como la de añadir un fieldset dinámicamente, obviaré esta última funcionalidad por haberla explicado en el ejercicio anterior.

La funcionalidad de múltiples pasos está implementada en las funciones: form_example_tutorial_10, form_example_tutorial_10_page_two y form_example_tutorial_10_submit. La primera de estas funciones form_example_tutorial_10 es como podéis imaginar la definición de formulario para la página 1 pero tiene además el siguiente if:

```
  if (isset($form_state['storage']['page_two'])) {
    return form_example_tutorial_10_page_two();
  }
```

form_example_tutorial_10_page_two tiene la definición del segundo paso del formulario. Y en form_example_tutorial_10_submit la variable $form_state['storage']['page_two']) se asigna a TRUE si se llama al botón submit cuyo id es 'next'.

4.11 Formulario para Subir un Fichero a Drupal

Este ejemplo permite al usuario subir un fichero a Drupal que es almacenado físicamente

y con una referencia en la base de datos.

```
function form_example_tutorial_11($form_state) {
  // This is required to upload files.
  // enctype="multipart/form-data" required by browsers to handle files.
  $form = array(
    '#attributes' => array('enctype' => "multipart/form-data"),
  );

  $form['file'] = array(
    '#type' => 'file',
    '#title' => t('Image'),
    '#description' => t('Upload a file, allowed extensions: jpg, jpeg, png, gif'),
  );

  $form['submit'] = array(
    '#type' => 'submit',
    '#value' => t('Submit'),
  );

  return $form;
}

/**
 * Valida el manejador para form_example_tutorial_11().
 * Verifica las extensiones válidas y verifica que el contenido es una imagen.
 */
function form_example_tutorial_11_validate($form, &$form_state) {
  $file = file_save_upload('file', array(
    'file_validate_extensions' => array('png gif jpg jpeg'),
    'file_validate_is_image' => array(),
  ));

  // Si el fichero pasó la validación:
  if (isset($file->filename)) {
    // Mueve el fichero dontro del sistema de ficheros Drupal
    if (file_move($file, $file->filename)) {
      // Actualiza la nueva localización de fichero en la base de datos.
      drupal_write_record('files', $file, 'fid');
      // Guarda el fichero para usarlo en el manejador de submit.
      $form_state['storage']['file'] = $file;
    }
    else {
      form_set_error('file', t('Failed to write the uploaded file the site\'s file folder.'));
    }
  }
  else {
    form_set_error('file', t('Invalid file, only images with the extension png, gif, jpg, jpeg are allowed'));
  }
}

/**
 * Manejador de submit para form_example_tutorial_11().
 */
function form_example_tutorial_11_submit($form, &$form_state) {
  $file = $form_state['storage']['file'];
  // Se termina con el archivo, se elimina su almacenamiento.
  unset($form_state['storage']['file']);
  // Se almacena el fichero de manera permanente
  file_set_status($file, FILE_STATUS_PERMANENT);
  // Se establece una respuesta para el usuario
  drupal_set_message(t('The form has been submitted and the image has been saved, filename:
@filename.', array('@filename' => $file->filename)));
```

```
}
```

La clave de este código está en las funciones form_example_tutorial_11_validate y form_example_tutorial_11_submit. En la primera se escribe en la base de datos como un fichero temporal con file_save_upload, seguidamente se escribe en el sistema de ficheros Drupal mediante file_move y se escribe en la tabla files mediante drupal_write_record. En form_example_tutorial_11_submit se elimina del alamacenamiento del form y se cambia el estado del fichero a FILE_STATUS_PERMANENT.

5. Mail API

Para conseguir una familiaridad con la API de correo de Drupal se desarrollará un módulo cuya funcionalidad sea enviar un correo desde un formulario. Obviamente esta funcionalidad se podría implementar en un sitio en producción mediante el módulo contact del core ó mediante el módulo webform. Sin embargo, aprender a hacer esto desarrollando un módulo permitirá el afianzar los conocimientos del API de form y es un ejemplo muy didáctico para aprender el API de correo. Aprender a hacer las cosas vía construir módulos sencillos permite aprender a leer y escribir módulos ajenos para poder resolver las necesidades específicas.

Del proyecto examples ya instalado, se activa el módulo E-mail example. Se puede echar un vistazo a email_example.module:

```
<?php
// $Id: email_example.module,v 1.1.2.4 2010/12/28 20:46:29 rfay Exp $

/**
 * @defgroup email_example Example: Email
 * @ingroup examples
 * @{
 * Ejemplo de mail API. (drupal 6)
 *
 * Este módulo de ejemplo proporciona 2 ejemplos diferentes del Drupal email
 * API.
 *  - define un simple formulario de contacto y muestra cómo usar drupal_mail()
 *    para enviar un e-mail (definido en hook_mail()) cuando el formulario es
 *    enviado.
 *  - muestra cómo los módulos pueden alterar emails definidos por otros
 *    módulos ó por el Core usando hook_mail_alter adjuntando una firma antes
 *    de enviarse
 *
 * Este ejemplo es parte del Examples for Developers Project que se puede
 * descargar y experimentar desde aquí: http://drupal.org/project/examples
 */

/**
 * Implementación de hook_mail().
 *
 * Este hook define una lista de posibles plantillas de correo que este módulo
 * puede enviar. Para cada correo hay un único identificador, ó 'key'.
 *
 * $message viene con algunas propiedades estándar ya definidas: dirección
 * 'to' , dirección 'from', y un conjunto de 'cabeceras' por defecto desde
 * drupal_mail(). El objetivo de hook_mail() es definir las propiedades
 * 'asunto' y 'cuerpo' del mensaje, así como crear cualquier ajuste en las
 * cabeceras que sea necesario
 *
 * El argumento $params es un array que puede manejar cualquier dato adicional
 * que se requiera para construir el asunto y cuerpo del mensaje; por ejemplo,
 * datos del formulario introducidos por el usuario, ó algún contexto de
```

```
 * información desde dónde el correo pueda venir.
 *
 * Nótese que hook_mail() actualmente no es un hook. Sólo es llamado para
 * un simple módulo, el módulo nombrado en el primer argumento de
 * drupal_mail(). Así es un callback de un tipo, pero no un hook.
 */
function email_example_mail($key, &$message, $params) {
  global $user;

  // Cada mensaje está asociado con un idioma, que puede ser ó no el idioma
  // seleccionado por el usuario actual, dependiendo del tipo de e-mail que
  // es enviado. Esta variable $language es usada más tarde en la llamada a t()
  // para el asunto y el cuerpo para asegurar que la apropiada traducción tiene
  // efecto.
  $language = $message['language'];

  switch ($key) {
    // Envía un mensaje simple desde el formulario de contacto
    case 'contact_message':
      $message['subject'] = t('E-mail sent from @site-name', array('@site-name' => variable_get('site_name',
'Drupal')), $language->language);
      // Nótese que el cuerpo del mensaje es un array, no una cadena de
      // carácteres.
      $message['body'][] = t('@name sent you the following message:', array('@name' => $user->name),
$language->language);
      // Debido a que esto es texto introducido por el usuario no se necesita
      // traducir.
      $message['body'][] = $params['message'];
      break;
  }
}

/**
 * Envía un correo.
 *
 * @param $form_values
 *   Un array de valores desde los campos del formulario de contacto que
 *   fueron enviados.
 *   Sólo hay dos ítems relevantes: $form_values['email'] y
 *   $form_values['message'].
 */
function email_example_mail_send($form_values) {
  // Todo sistema de correo necesita especificar el módulo y el identificador
  // de plantilla
  $module = 'email_example';
  $key = 'contact_message';

  $to = $form_values['email'];
  $from = variable_get('site_mail', 'admin@example.com');

  // "params" carga en un contexto adicional para completar el contenido de
  // contenido de correo en hook_mail(). En este caso, se quieren pasar los
  // valores que el usuario introdujo en el formulario, que incluye el cuerpo
  // del mensaje en $form_values['message'].
  $params = $form_values;

  // El idioma del correo. Se usará el idoma por defecto del sitio
  $language = language_default();

  // Si el correo se manda ó no automáticamente cuando drupal_mail() es
  // llamado. Por defecto es TRUE, y normalmente es lo que se quiere a menos
  // que se necesite hacer procesamiento adicional antes que drupal_mail_send()
```

```
  // es llamado.
  $send = TRUE;
  // Envía el correo y chequea el éxito del envío. Nótese que esto no
  // garantiza la entrega del correo; sólo que no hubo cuestiones de PHP
  // encontradas mientras se enviaba
  $result = drupal_mail($module, $key, $to, $language, $params, $from, $send);
  if ($result['result'] == TRUE) {
    drupal_set_message(t('Your message has been sent.'));
  }
  else {
    drupal_set_message(t('There was a problem sending your message and it was not sent.'), 'error');
  }

}

/**
 * Implementación de hook_mail_alter().
 *
 * Esta función no es necesaria para enviar un correo usando el sistema
 * de correo de Drupal.
 *
 * Hook_mail_alter() proporciona un interfaz para alterar cualquier aspecto
 * de un correo enviado por Drupal. Se puede usar este hook para añadir un
 * pie común a todo el correo saliente, añadir campos de cabecera extra, y/o
 * modificar el correo en cualquier modo. Convertir a HTML el correo saliente
 * es una posibilidad.
 */
function email_example_mail_alter(&$message) {
  // Para el propósito de este ejemplo, modifica todos los mensajes de correo
  // saliente y adjunta una firma. La firma será traducida al idioma en el que
  // el mensaje fue construido.
  $signature = t("\n--\nMail altered by email_example module.", array(), $message['language']->language);
  if (is_array($message['body'])) {
    $message['body'][] = $signature;
  }
  else {  // Algunos módulos usan el cuerpo como una cadena de caracteres,
          // erróneamente.
    $message['body'] .= $signature;
  }
}

///// Funciones de Soporte ////

/**
 * Implementación de hook_menu().
 *
 * Configura una página con un formulario de contacto vía correo dentro.
 */
function email_example_menu() {
  $items['example/email_example'] = array(
    'title' => 'E-mail Example: contact form',
    'page callback' => 'drupal_get_form',
    'page arguments' => array('email_example_form'),
    'access arguments' => array('access content'),
  );

  return $items;
}

/**
 * El formulario de contacto.
 */
```

```
function email_example_form() {
  $form['intro'] = array(
    '#value' => t('Use this form to send a message to an e-mail address. No spamming!'),
  );
  $form['email'] = array(
    '#type' => 'textfield',
    '#title' => t('E-mail address'),
    '#required' => TRUE,
  );
  $form['message'] = array(
    '#type' => 'textarea',
    '#title' => t('Message'),
    '#required' => TRUE,
  );
  $form['submit'] = array(
    '#type' => 'submit',
    '#value' => t('Submit'),
  );

  return $form;
}

/**
 * Lógica de validación del formulario.
 */
function email_example_form_validate($form, &$form_state) {
  if (!valid_email_address($form_state['values']['email'])) {
    form_set_error('email', t('That e-mail address is not valid.'));
  }
}

/**
 * Lógica después del envío del formulario.
 */
function email_example_form_submit($form, &$form_state) {
  email_example_mail_send($form_state['values']);
}

/**
 * @} End of "defgroup email_example".
 */
```

Si se fijan en la implementación de hook_menu (email_example_menu) se ve que la ruta para visualizar este módulo es example/email_example que es introducida en el navegador.

Otras funciones como email_example_form, email_example_form_validate y email_example_form_submit también son hooks cuya funcionalidad ya se ha visto en el apartado Form API y que son, por tanto, para la construcción del formulario.

Las funciones que introducen a los nuevos hooks de la API de correo son: email_example_mail (hook_mail), email_example_mail_alter (hook_mail_alter) y una función auxiliar email_example_mail_send que llama a drupal_mail. Ahora se estudiará el comportamiento de cada una.

email_example_mail define el asunto ($message['subject']) y el cuerpo ($message['body'][]) del mensaje. Mediante el último argumento a la función t se define el idioma.

email_example_mail_send establece el valor de todos los argumentos que va a necesitar **drupal_mail** para ejecutarse. Estos valores son en su mayoría tomados de los

valores introducidos por el usuario en el formulario.

email_example_mail_alter. El hook hook_mail_alter permite modificar aspectos del mensaje. La implementación del hook ha sido realizada por motivos didácticos y añade una firma al mensaje. Obviamente esto podría haberse introducido dentro de email_example_mail.

Para una mejor comprensión y una información más detallada se estudiará la documentación de la API de drupal_mail.

5.1 drupal_mail

drupal_mail($module, $key, $to, $language, $params = array(), $from = NULL, $send = TRUE)
Compone y opcionalmente envía un mensaje.

El envío de correo funciona definiendo una plantilla (asunto, texto y posibles cabeceras de e-mail) y los valores de reemplazo para usar en los lugares apropiados de la plantilla. Dichas plantillas se construyen desde el hook_mail() que hay en el módulo que envía el correo. Cualquier módulo puede modificar el mensaje de correo usando hook_mail_alter(). Finalmente, drupal_mail_system()->mail() envía el correo, que puede ser reutilizado si ese correo fue compuesto para ser enviado a múltiples destinos.

Encontrar en qué idioma se enviará un correo necesita alguna consideración. Si se envía correo a un usuario, su idioma preferido sería lo ideal, así se puede usar la función user_preferred_language(). Si se envía correo según los valores rellenados en un formulario de una determinada página, hay dos elecciones que hay que hacer, a menos que estés enviando el correo a un usuario del sitio. Se puede usar el lenguaje usado para generar la página (la variable global $language) o el lenguaje por defecto del sitio (ver language_default()). El primer método es bueno para enviar correo a la persona que está rellenando el formulario y el segundo es bueno si se envía el correo a una dirección previamente configurada (como la dirección de contacto).

Parámetros

$module Un módulo invoca a hook_mail(). El hook {$module}_mail() será llamado para completar la estructura de $message que ya contiene cierto valores por defecto.

$key Una clave identifica el correo enviado. El id del correo para alterarlo será {$module}_{$key}.

$to La dirección ó direcciones de correo a las que el mensaje se enviará. El formato de esta cadena cumplirá con el RFC 2822. Algunos ejemplos son:

- user@example.com
- user@example.com, anotheruser@example.com
- User <user@example.com>
- User <user@example.com>, Another User <anotheruser@example.com>

$language El idioma usado para componer el correo.

$params Parámetro opcionales para construir el correo.

$from Establece el From si este es dado.

$send Envía el mensaje directamente, sin llamar a drupal_mail_system()->mail() manualmente.

Valor devuelto

La estructura del array $message conteniendo todos los detalles del mensaje. Si ya se envió ($send = TRUE), entonces el elmento 'result' contendrá el indicador de éxitos del correo, si ha habido fallo se escribirá en el watchdog. (Éxito solo significa que el mensaje ha sido aceptado a nivel php, lo que no garantiza que sea entregado.)

6. Creando Tipos de Contenido

Ahora se va a explicar un ejemplo de cómo un módulo puede ser usado para definir un nuevo tipo de nodo. Decir que los tipos de nodo personalizados son a menudo creados

con CCK.

Para poder visualizar el ejemplo es necesario activar el módulo “Node example” del proyecto “Examples” (http://drupal.org/project/examples).

Como se puede comprobar el tipo de nodo que se crea permite a los usuarios especificar un color (*“color”)* y una cantidad (*“quantity”*).

6.1 Crear la estructura en la base de datos

Para almacenar esta información extra, se necesita una tabla de base de datos auxiliar definida en node_example_schema(). Se puede estudiar dicho código abriendo el fichero node_example.install:

```
<?php
// $Id: node_example.install,v 1.1.2.1 2009/10/14 23:51:34 jhodgdon Exp $

/**
 * @file
 * Código para instalar y desinstalar el módulo Node example.
 */

/**
 * Implementación de hook_install().
 */
function node_example_install() {
  drupal_install_schema('node_example');
}

/**
 * Implementación de hook_uninstall().
 */
function node_example_uninstall() {
  drupal_uninstall_schema('node_example');
}

/**
 * Implementación de hook_schema().
 */
function node_example_schema() {
  $schema['node_example'] = array(
    'fields' => array(
      'vid'      => array('type' => 'int', 'unsigned' => TRUE, 'not null' => TRUE, 'default' => 0),
      'nid'      => array('type' => 'int', 'unsigned' => TRUE, 'not null' => TRUE, 'default' => 0),
      'color'    => array('type' => 'varchar', 'length' => 255, 'not null' => TRUE, 'default' => ''),
      'quantity' => array('type' => 'int', 'unsigned' => TRUE, 'not null' => TRUE, 'default' => 0),
    ),
    'primary key' => array('vid', 'nid'),
  );

  return $schema;
}
```

Como se puede observar lo más interesante es la creación del array donde se define el tipo de cada nuevo campo en el nuevo tipo de contenido.

6.2 hook_menu

Ahora se va estudiar función por función el fichero node_example.module. Empezando por node_example_menu():

```
function node_example_menu() {
  $items['examples/node_example'] = array(
    'title' => 'Node Example',
    'page callback' => 'node_example_info',
    'access callback' => TRUE,
  );
  return $items;
}

/**
 * Explica cómo el módulo muestra un nuevo tipo de nodo
 */
function node_example_info() {
            return t('The node example defines a new node type, "Example node type 1", which can be created
at !link.', array('!link' => l(t('node/add/example-node-type-1'), 'node/add/example-node-type-1')));
}
```

Este hook establece que en la ruta examples/node_example aparecerá el resultado de node_example_info. Lo cual no es más que un texto informando de cómo crear un nuevo nodo de este tipo.

6.3 hook_node_info

```
function node_example_node_info() {
  return array(
    'example_node_type_1' => array(
      'name' => t('Example node type 1'),
      'module' => 'node_example',
      'description' => t("An example node type with a few fields."),
      'has_title' => TRUE,
      'title_label' => t('Example Node Type 1 Title'),
      'has_body' => TRUE,
      'body_label' => t('Example Node Type 1 Body'),
    )
  );
}
```

El hook_node_info devuelve un array con información obligatoria y opcional del tipo de contenido. Nótese que el nombre máquina del tipo de contenido será example_node_type_1, el resto es autoexplicativo.

6.4 hook_access

```
function node_example_access($op, $node, $account) {
  if ($op == 'create') {
    return user_access('create example content', $account);
  }

  if ($op == 'update') {
    if (user_access('edit any example content', $account) || (user_access('edit own example content',
$account) && ($account->uid == $node->uid))) {
      return TRUE;
    }
  }

  if ($op == 'delete') {
    if (user_access('delete any example content', $account) || (user_access('delete own example content',
$account) && ($account->uid == $node->uid))) {
      return TRUE;
```

```
    }
  }
}
```

hook_access define restricciones de acceso a los tipos de contenido que los define.

6.5 hook_perm

```
function node_example_perm() {
  return array(
    'create example content',
    'delete own example content',
    'delete any example content',
    'edit own example content',
    'edit any example content',
  );
}
```

hook_perm define los posibles permisos que va a tener el tipo de contenido.

6.6 hook_form

```
function node_example_form(&$node, $form_state) {
  // El admin del sitio puede decidir si este tipo de nodo tiene un título y
  // un cuerpo, y cómo los campos son etiquetados. Se necesita cargar estas
  // configuraciones, así se podrá construir el formulario del nodo
  // correctamente.
  $type = node_get_types('type', $node);

  if ($type->has_title) {
    $form['title'] = array(
      '#type' => 'textfield',
      '#title' => check_plain($type->title_label),
      '#required' => TRUE,
      '#default_value' => $node->title,
      '#weight' => -5
    );
  }

  if ($type->has_body) {
    // En Drupal 6, se usa node_body_field() para obtener el cuerpo y filtrar
    // elementos. Esto reemplaza el antiguo método de textarea + filter_form()
    // de configurarlo. También asegura que la división del teaser se hace
    // apropiadamente.
    $form['body_field'] = node_body_field($node, $type->body_label, $type->min_word_count);
  }

  // Ahora se definen los elementos de formulario específico para el tipo de
  // nodo.
  $form['color'] = array(
    '#type' => 'textfield',
    '#title' => t('Color'),
    '#default_value' => isset($node->color) ? $node->color : '',
  );
  $form['quantity'] = array(
    '#type' => 'textfield',
    '#title' => t('Quantity'),
    '#default_value' => isset($node->quantity) ? $node->quantity : 0,
    '#size' => 10,
    '#maxlength' => 10
  );
```

```
  return $form;
}

/**
 * Implementación de hook_validate().
 *
 * El campo "quantity" requiere un número para ser introducido. Este hook
 * permite asegurar que usuario introdujo un valor apropiado antes que se
 * intente insertar cualquier cosa en la base de datos.
 *
 * Se enviará una señal de los errores con form_set_error().
 */
function node_example_validate($node, &$form) {
  if ($node->quantity) {
 b
    if (!is_numeric($node->quantity)) {
      form_set_error('quantity', t('The quantity must be a number.'));
    }
  }
}
```

Como ya se ha visto en anteriores secciones hook_form define el formulario y hook_validate valida sus campos.

6.7 hook_insert

```
function node_example_insert($node) {
  db_query("INSERT INTO {node_example} (vid, nid, color, quantity) VALUES (%d, %d, '%s', %d)", $node->vid, $node->nid, $node->color, $node->quantity);
}
```

hook_insert, inserción de un nuevo nodo.

6.8 hook_update

```
function node_example_update($node) {
  // si este es un nuevo nodo o se está añadiendo una nueva revisión
  if ($node->revision) {
    node_example_insert($node);
  }
  else {
    db_query("UPDATE {node_example} SET color = '%s', quantity = %d WHERE vid = %d", $node->color, $node->quantity, $node->vid);
  }
}
```

hook_update, actualiza un nodo ya creado.

6.9 hook_nodeapi

```
function node_example_nodeapi(&$node, $op, $teaser, $page) {
  switch ($op) {
    case 'delete revision':
      // Notice that we're matching a single revision based on the node's vid.
      db_query('DELETE FROM {node_example} WHERE vid = %d', $node->vid);
      break;
  }
}
```

Para eliminar una revisión solo es posible hacerlo usando hook_nodeapi. Decir que las

funciones de inserción y actualización se podrían haber implementado con este hook como ya se verá más adelante.

6.10 hook_delete

```
function node_example_delete($node) {
  // Nótese que estamos seleccionando todas las revisiones, al usar el nid de
  // nodo
  db_query('DELETE FROM {node_example} WHERE nid = %d', $node->nid);
}
```

hook_delete, elimina un nodo de la base de datos.

6.11 hook_load

```
function node_example_load($node) {
  $additions = db_fetch_object(db_query('SELECT color, quantity FROM {node_example} WHERE vid =
%d', $node->vid));
  return $additions;
}
```

Al hacer el node_load es necesario que cargue también los campos específicos de ese tipo de contenido, para ello existe hook_load.

6.12 hook_view

```
function node_example_view($node, $teaser = FALSE, $page = FALSE) {
  $node = node_prepare($node, $teaser);
  $node->content['myfield'] = array(
    '#value' => theme('node_example_order_info', $node),
    '#weight' => 1,
  );

  return $node;
}
```

hook_view es el encargado de la visualización del nodo. En este caso se ha añadido un nuevo campo “myfield” que es el responsable de mostrar la información extra del nodo: “*The order is for x y items.*”. Y cuyo contenido es definido en la función theme_node_example_order_info que se verá más adelante.

6.13 hook_theme

```
function node_example_theme() {
  return array(
    'node_example_order_info' => array(
      'arguments' => array('node'),
    ),
  );
}
```

hook_theme define las funciones theme y sus argumentos. En este caso node_example_order_info es la función theme.

```
function theme_node_example_order_info($node) {
  $output = '<div class="node_example_order_info">';
  $output .= t('The order is for %quantity %color items.', array('%quantity' => check_plain($node->quantity),
'%color' => check_plain($node->color)));
  $output .= '</div>';
  return $output;
```

```
}
```

Devuelve el html que se visualizará.

7. Creación de Menús

En esta sección se estudiarán los hooks: hook_menu(), hook_menu_alter() y hook_menu_link_alter().

En anteriores ejemplos ya se ha visto que hook_menu sirve para establecer un contenido en una url. Ahora se verá como usar hook_menu para establecer menús, permisos, tabs, títulos dinámicos, etc.

Para ello se utilizará de nuevo el proyecto Examples (http://drupal.org/project/examples) activando el módulo “Menu Example”.

Ahora se va mostrar uno a uno los ítems del hook_menu del menu_example.module:

7.1 Un ítem simple

```
$items['menu_example'] = array(
  // El tipo de menú no es obligatorio, aquí se está usando el de por
  // defecto.
  // 'type' => MENU_NORMAL_ITEM,

  // El título – NO usa t(), ya que t() es llamado automáticamente.
  'title' => 'Menu Example',

  // Descripción – NO usa t(), ya que t() es llamado automáticamente.
  'description' => 'Simplest possible menu type, and the parent menu entry for others',

  // Función que es llamada cuando la ruta es accedida.
  'page callback' => '_menu_example_basic_instructions',

  // Argumentos para el page callback. Aquí se usarán para dar contenido a
  // la página.
  'page arguments' => array(t('This page is displayed by the simplest (and base) menu example. Note that the title of the page is the same as the link title. You can also <a href="!link">visit a similar page with no menu link</a>', array('!link' => url('menu_example/path_only')))),

  // Esto está para ser accesible a todos los usuarios, así 'access callback'
  // puede ser asignado a TRUE, lo que significa que debemos pasar por alto
  // todos los controles de acceso
  'access callback' => TRUE,

  // Si el page callback está localizado en otro fichero, especifícalo aquí
  // y este fichero será cargado automáticamente cuando se necesite.
  // 'file' => 'menu_example.module',

  // Se puede elegir qué menú tiene el enlace. Por defecto es 'navigation'.
  // 'menu_name' => 'navigation',

  // Muestra el enlace de menú expandido.
  'expanded' => TRUE,
);
```

De este modo, en la ruta menu_example se encuentra el resultado de la función _menu_example_basic_instructions. Como ítem de menú el título es “Menu Example”, la descripción “Simplest possible menu type, and the parent menu entry for others”, aparece activado, expandido y su menú es “navigation” puesto que no está definido.

7.2 Un ítem en un menú alternativo

```
  $items['menu_example_alternate_menu'] = array(
    'title' => 'Menu Example: Menu in alternate menu',

    // Nombre máquina del menú en el que el enlace aparecerá.
    'menu_name' => 'primary-links',

    'page callback' => '_menu_example_menu_page',
    'page arguments' => array(t('This will be in the Primary Links menu instead of the default Navigation
menu')),
    'access callback' => TRUE,
  );
```

Muestra un ítem de menú, en un menú distinto a “Navigation”. Esto se hace mediante la clave 'menu_name'.

7.3 Un ítem con restricciones de permisos

Ahora se verán 2 entradas de menú: una es la ruta para entrar en la otra. Siendo esta última una entrada con restricciones de permisos.

```
  // Una entrada de menú con permisos usando user_access().

  // Primero, proporciona un menu ítem de cortesía que menciona la existencia
  // del ítem con permisos.
  $items['menu_example/permissioned'] = array(
    'title' => 'Permissioned Example',
    'page callback' => '_menu_example_menu_page',
    'page arguments' => array(t('A menu item that requires the "access protected menu example" permission
is at <a href="!link">menu_example/permissioned/controlled</a>', array('!link' =>
url('menu_example/permissioned/controlled')))),
    'access callback' => TRUE,
    'expanded' => TRUE,
  );

  // Ahora proporciona el menú ítem con permisos.
  $items['menu_example/permissioned/controlled'] = array(
    'title' => 'Permissioned Menu Item',
    'description' => 'This menu entry will not show and the page will not be accessible without the "access
protected menu example" permission.',
    'page callback' => '_menu_example_menu_page',
    'page arguments' => array(t('This menu entry will not show and the page will not be accessible without
the "access protected menu example" permission.')),

    // Para una entrada de menú con permisos, se proporcio un access callback
    // que determina si el usuario actual tiene acceso. Por defecto es
    // user_access(), que es el que se usará en este caso. Como es el caso por
    // defecto no hay que introducirlo.
    // 'access callback' => 'user_access',

    // Los 'access arguments' son pasado al 'access callback' para ayudarle a
    // a hacer su trabajo. En el caso de user_access(), se necesita pasar un
    // permiso como primer argumento.
    'access arguments' => array('access protected menu example'),

    // El elemento opcional weight habla acerca del orden de los ítems de
    // submenú. Pesos altos, llevan los ítems abajo del menú.
    'weight' => 10,
  );
```

La clave 'access arguments' dice que permisos tiene que tener el usuario para poder visualizar el ítem.

Estos permisos han sido definidos vía hook_perm más adelante en el código:

```
/**
 * Implementa hook_perm() para proporciona una demostración de cadena de
 * acceso.
 */
function menu_example_perm() {
  return array('access protected menu example');
}
```

7.4 Un ítem que no es un menu link

De nuevo, 2 ítems: uno de cortesía para entrar en el otro. El segundo no es accesible desde el menú navigation, ni ningún otro.

```
  // Un ítem de menú, sin enlace en el menú. Esto puede se usado en cualquier
  // momento que no se precise que el usuario vea un enlace en el menú. Por
  // otro lado, es lo mismo que la entrada más simple. MENU_CALLBACK es usado
  // para todos los ítems de menú que no necesitas un enlace de menú visible,
  // incluyendo servicios y otras páginas que pueden ser enlazados, pero que
  // no se pretende acceder directamente.

  // Primero, se proporciona un enlace de cortesía para que la gente pueda
  // encontrarlo.
  $items['menu_example/path_only'] = array(
    'title' => 'MENU_CALLBACK example',
    'page callback' => '_menu_example_menu_page',
    'page arguments' => array(t('A menu entry with no menu link (MENU_CALLBACK) is at <a href="!link">!
link</a>', array('!link' => url('menu_example/path_only/callback')))),
    'access callback' => TRUE,
    'weight' => 20,
  );
  $items['menu_example/path_only/callback'] = array(

    // Elegir el tipo MENU_CALLBACK significa dejar la ruta completamente
    // fuera de los enlaces de menú.
    'type' => MENU_CALLBACK,

    // El título se usa para el título de la página, aunque no sea usado
    // como texto para el enlace de menú, ya que no hay enlace de menú.
    'title' => 'Callback Only',

    'page callback' => '_menu_example_menu_page',
    'page arguments' => array(t('The menu entry for this page is of type MENU_CALLBACK, so it provides
only a path but not a link in the menu links, but it is the same in every other way to the simplest example.')),
    'access callback' => TRUE,
  );
```

El ítem es del tipo MENU_CALLBACK que define no ser accesible desde menú.

7.5 Un ítem de menú con tabs

Para tener tabs son necesarias al menos 3 cosas:

1. Un padre ítem de menú del tipo MENU_NORMAL_ITEM (menu_example/tabs en este ejemplo)
2. Un tab primario (el único que está activo cuando se entra en el menú). Este tab es

del tipo MENU_DEFAULT_LOCAL_TASK

3. Algunas otras entradas de menú para los otros tabs, serán del tipo MENU_LOCAL_TASK

Ahora se estudiará el código:

```
  $items['menu_example/tabs'] = array(
    // 'type' => MENU_NORMAL_ITEM,  // No necesario, ya que es el valor por
                                    // defecto.
    'title' => 'Tabs',
    'description' => 'Shows how to create primary and secondary tabs',
    'page callback' => '_menu_example_menu_page',
    'page arguments' => array(t('This is the "tabs" menu entry.')),
    'access callback' => TRUE,
    'weight' => 30,
  );
  $items['menu_example/tabs/default'] = array(
    'type' => MENU_DEFAULT_LOCAL_TASK,
    'title' => t('Default primary tab'),
    'weight' => 1,
  );
  // Ahora se añade el resto de las entradas tab.
  foreach(array(t('second') => 2, t('third') => 3, t('fourth') => 4) as $tabname => $weight) {
    $items["menu_example/tabs/$tabname"] = array(
      'type' => MENU_LOCAL_TASK,
      'title' => $tabname,
      'page callback' => '_menu_example_menu_page',
      'page arguments' => array(t('This is the tab "@tabname" in the "basic tabs" example', array('@tabname'
=> $tabname))),
      'access callback' => TRUE,

      // La propiedad peso sobreescribe el orden alfabético (por defecto) de las
      // entradas del menú, permitiendo obtener los tabs en el orden deseado.
      'weight' => $weight,
    );
  }

  // Finalmente, se añaden tabs secundarios al tab por defecto
  $items['menu_example/tabs/default/first'] = array(
    'type' => MENU_DEFAULT_LOCAL_TASK,
    'title' => t('Default secondary tab'),
    // La página de callback adicional y los ítems relacionado son manjejados
    // por el ítem de menú padre
  );
  foreach(array(t('second'), t('third')) as $tabname) {
    $items["menu_example/tabs/default/$tabname"] = array(
      'type' => MENU_LOCAL_TASK,
      'title' => $tabname,
      'page callback' => '_menu_example_menu_page',
      'page arguments' => array(t('This is the secondary tab "@tabname" in the "basic tabs" example
"default" tab', array('@tabname' => $tabname))),
      'access callback' => TRUE,
    );
  }
```

7.6 Ítem de menú con argumentos extra

```
  $items['menu_example/use_url_arguments'] = array(
    'title' => 'Extra Arguments',
    'description' => 'The page callback can use the arguments provided after the path used as key',
    'page callback' => '_menu_example_menu_page',
    'page arguments' => array(t('This page demonstrates using arguments in the path (portions of the path
after "menu_example/url_arguments". For example, access it with <a href="!link1">!link1</a> or <a href="!
link2">!link2</a>).', array('!link1' => url('menu_example/use_url_arguments/one/two'), '!link2' =>
url('menu_example/use_url_arguments/firstarg/secondarg')))),
    'access callback' => TRUE,
    'weight' => 40,
  );
```

Como se puede observar la definición del ítem no es muy distinta a la de un ítem simple. El tratamiento de argumentos viene dado en el callback _menu_example_menu_page:

```
function _menu_example_menu_page($content = NULL, $arg1 = NULL, $arg2 = NULL) {
  $output =  '<div>' . $content . '</div>';

  if (!empty($arg1)) {
    $output .= '<div>' . t('Argument 1=%arg', array('%arg' => $arg1)) . '</div>';
  }
  if (!empty($arg2)) {
    $output .= '<div>' . t('Argument 2=%arg', array('%arg' => $arg2)) . '</div>';
  }
  return $output;
}
```

7.7 Ítem con el título generado dinámicamente

```
  $items['menu_example/title_callbacks'] = array(
    'title callback' => '_menu_example_simple_title_callback',
    'title arguments' => array(t('Dynamic title: username=')),
    'description' => 'The title of this menu item is dynamically generated',
    'page callback' => '_menu_example_menu_page',
    'page arguments' => array(t('The menu title is dynamically changed by the title callback')),
    'access callback' => TRUE,
    'weight' => 50,
  );
```

El título de menú puede ser dinámicamente creado usando el 'title callback' que por defecto es t() y en este caso llama a la siguiente función:

```
function _menu_example_simple_title_callback($base_string) {
  global $user;
  $username = !empty($user->name) ? $user->name : t('anonymous');
  return $base_string . ' ' . $username;
}
```

7.8 Capturando argumentos de la url

En ocasiones va a interesar capturar argumentos de la url, por ejemplo, un id. Para ilustrar cómo hacer esto se van a crear 2 ítems de menú: uno de entrada y el siguiente donde se desarrolla la acción.

```
  $items['menu_example/placeholder_argument'] = array(
    'title' => 'Placeholder Arguments',
    'page callback' => '_menu_example_menu_page',
    'page arguments' => array(t('Demonstrate placeholders by visiting <a href="!
link">menu_example/placeholder_argument/3343/display</a>', array('!link' =>
```

```
url('menu_example/placeholder_argument/3343/display'))),
    'access callback' => TRUE,
    'weight' => 60,
  );

  // Ahora la entrada actual
  $items['menu_example/placeholder_argument/%/display'] = array(
    'title' => 'Placeholder Arguments',
    'page callback' => '_menu_example_menu_page',
    'page arguments' => array(2),
    'access callback' => TRUE,
  );
```

page_arguments está cogiendo el 2º argumento (siendo 0 el primero). De este modo, si se prueba a cambiar el 2 por 3 la página devolverá display.

7.9 Capturando un argumento opcional

En el caso anterior el argumento era obligatorio, pero también se puede introducir un argumento opcional. En el siguiente código se ve cómo:

```
  $items['menu_example/default_arg/%menu_example_arg_optional'] = array(
    'title' => 'Processed Placeholder Arguments',
    'page callback' => '_menu_example_menu_page',
    'page arguments' => array(2),  // arg 2 (3rd arg) is the one we want.
    'access callback' => TRUE,
    'weight' => 70,
  );
```

Nótese que esta vez se pone % seguido de menu_example_arg_optional esto implica que se llama a una función con dicho nombre. La función es:

```
function menu_example_arg_optional_load($id) {
  $mapped_value = _menu_example_mappings($id);
  if (!empty($mapped_value)) {
    return t('Loaded value was %loaded', array('%loaded' => $mapped_value));
  }
  else {
    return t('Sorry, the id %id was not found to be loaded', array('%id' => $id));
  }
}
```

Y menu_example_mappings se define:

```
function _menu_example_mappings($id) {
  $mapped_value = NULL;
  static $mappings = array(
    1 => 'one',
    2 => 'two',
    3 => 'three',
    99 => 'jackpot! default',
  );
  if (isset($mappings[$id])) {
    $mapped_value = $mappings[$id];
  }
  return $mapped_value;
}
```

7.10 hook_menu_alter

Cuando se despliegan sitios web en Drupal, puede ser necesario modificar ítems de menús ya definidos en Drupal. Esto va a ser posible mediante hook_menu_alter, en el siguiente ejemplo se ilustra cómo:

```
function menu_example_menu_alter(&$items) {
  // Change the path 'logout' to a Spanish 'salir".
  if (!empty($items['logout'])) {
    $items['salir'] = $items['logout'];
    unset($items['logout']);
  }

  if (!empty($items['user/%user_uid_optional'])) {
    $items['user/%user_uid_optional']['title callback'] = 'menu_example_user_page_title';
  }
}
```

En el primer if se activa la ruta 'salir' que va a tener la misma funcionalidad que hasta ahora tenía logout, ya que éste se desactiva.
El segundo if define para el siguiente title callback para user:

```
function menu_example_user_page_title($account) {
  return t("@name's account", array('@name' => $account->name));
}
```

7.11 hook_menu_link_alter

hook_menu_link_alter permite modificar datos que han sido guardados en la tabla menu_links. Un ejemplo a estudiar es:

```
function menu_example_menu_link_alter(&$item, $menu) {
  // Force the link title to remain 'Clear Cache' no matter what the admin
  // does with the web interface.
  if ($item['link_path'] == 'devel/cache/clear') {
     $item['link_title'] = 'Clear Cache';
  };
}
```

En este ejemplo se modifica el título del ítem que enlaza a devel/cache/clear.

8. Forums: un módulo de la vida real

Hasta ahora se han estudiado módulos que tenían un alto valor pedagógico, pero escaso valor funcional. Sin embargo, se han establecido las bases para poder comprender cualquier módulo de la vida real. A modo de ejemplo se va a hacer una lectura comentada del módulo forums que es un módulo del core de Drupal. Los módulos del core garantizan una calidad de código bastante alta. Cuando se desarrollen nuevos módulos será importante fijarse en los módulos del core para asegurar que se están haciendo bien las cosas.

Para poder seguir esta exposición es necesario activar el módulo forum. Y acceder al módulos desde modules/forum.

Ahora se va a ir explicando el código de forum desde los conceptos ya aprendidos en las secciones ya vistas. De tal manera, que servirá de repaso e integración de todo lo aprendido.

8.1 Forum implementa un tipo de contenido

En realidad, el tipo de contenido es el tema del foro y tanto los foros como los contenedores son terms de una taxonomía. Es importante tener en cuenta estos conceptos antes de comenzar

Como tipo de contenido realiza las siguientes acciones:

1. *Instala el esquema de base de datos*: ver forum.install
2. *forum_term_load*: cada foro es un term, por lo que es necesario tenerlo en cuenta
3. *forum_nodeapi*: implementa las acciones para los “temas de foro” (view, prepare, validate, presave, update, insert, delete y load)
4. *forum_node_info*: define el tipo de contenido.
5. *forum_perm*: define los permisos existentes
6. *forum_access*: define qué permisos definidos son necesarios para acceder a cada una de las acciones de nodeapi.
7. *forum_form_alter*: modifica taxonomy_form_vocabulary, taxonomy_form_term y forum_node_form
8. *forum_form*: el title y el body es similar node_example, añade el campo shadow.
9. *forum_load*: carga el nodo

8.2 El hook_menu de forum

Define tanto el ítem de menú forum como las páginas de administración. Nótese que no es necesario definir los ítems de node/add, node/1, ya que son propios de node.

8.3 Dos bloques para forum

Si se va a la administración de bloques se puede observar que es posible añadir los bloques de temas activos y temas nuevos. Estos bloques tiene la pecularidad frente a lo ya estudiado que es posible configurarlos y guardar los cambios hechos. Dejo como ejercicio al lector que viendo el código de forums (forum_block) intente implementar dichas funcionalidades al módulo de la sección “Creación de Bloques”.

8.4 Plantillas para forum

Para modificar aspectos de visualización en la mayoría de los CMS existentes se usa el concepto de plantilla (en inglés *template*). Viendo el directorio forum se pueden detectar un buen número de ficheros con la extensión tpl.php éstos son todas las plantillas que forum tiene. Es posible que no guste el aspecto en que forums organiza el html ó la css. En ese caso habría que reescribir los ficheros *.tpl.php ó el fichero forum.css en el directorio del tema. En el próximo capítulo se verá cómo modificar ficheros tpl.php, si bien es bastante intuitivo.

El mecanismo para facilitar el trabajo con plantillas depende de hook_theme y template_preprocess. hook_theme define las plantillas existentes, mientras que template_preprocess define las variables con las que se va a poder trabajar en estas plantillas.

En forum se implementa hook_theme del siguiente modo:

```
function forum_theme() {
```

```
  return array(
    'forums' => array(
      'template' => 'forums',
      'arguments' => array('forums' => NULL, 'topics' => NULL, 'parents' => NULL, 'tid' => NULL, 'sortby' =>
NULL, 'forum_per_page' => NULL),
    ),
    'forum_list' => array(
      'template' => 'forum-list',
      'arguments' => array('forums' => NULL, 'parents' => NULL, 'tid' => NULL),
    ),
    'forum_topic_list' => array(
      'template' => 'forum-topic-list',
      'arguments' => array('tid' => NULL, 'topics' => NULL, 'sortby' => NULL, 'forum_per_page' => NULL),
    ),
    'forum_icon' => array(
      'template' => 'forum-icon',
      'arguments' => array('new_posts' => NULL, 'num_posts' => 0, 'comment_mode' => 0, 'sticky' => 0),
    ),
    'forum_topic_navigation' => array(
      'template' => 'forum-topic-navigation',
      'arguments' => array('node' => NULL),
    ),
    'forum_submitted' => array(
      'template' => 'forum-submitted',
      'arguments' => array('topic' => NULL),
    ),
  );
}
```

8.5 Otras cuestiones

Aún hay algunas funciones que no han sido explicadas como: _forum_user_last_visit, _forum_get_topic_order, _forum_get_topic_order_sql, _forum_new, _forum_topics_unread que son funciones que devuelven el resultado de una consulta sql.
Con lo que se ha visto hasta ahora ya se es posible entender la mayoría de los módulos y así poder adaptar Drupal a necesidades particulares, por ejemplo, creando módulos con personalizaciones ó incluyendo éstas en el propio tema.

9. Ejercicios Prácticos

Ahora el alumno/a está listo para implementar sus propios módulos poniendo en práctica los conocimientos aprendidos.

9.0 Conocimientos de hooks

Describe con tus propias palabras qué hace: hook_menu, hook_mail, hook_node_info, hook_perm, hook_access, hook_node_info, hook_insert, hook_update, hook_update, hook_delete, hook_load, hook_view, hook_theme

9.1 Hello World

Escribir un módulo que implemente una página que implemente la frase “Hello World”, debe ser accesible por la url /hello-world y accesible desde el menú primary-links.

9.2 Bloque más consulta

Implementar un bloque mediante la api de drupal que muestre los nodos ordenados por número de comentarios.

9.3 Producto

Crear un módulo que implemente el tipo de contenido Producto. Un producto tiene un título, una imagen, un precio y un texto. Cuantos más hooks se demuestre que se conoce mejor se evaluará el ejercicio.

9.4 Formulario de tres pasos

Crear un módulo completo que implemente un formulario de tres pasos:

1) Nombre, Apellidos y Teléfono de contacto. El teléfono es obligatorio y debe empezar con 91.
2) Correo electrónico, Motivo del contacto.
3) Cuerpo del mensaje.

El módulo además manda un correo electrónico al mail del admin de la instalación de Drupal.

9.5 Blog

Crear un módulo que implemente el tipo de contenido Blog. Este módulo deberá tener un formulario (al igual que Node Example). Y se enviará un correo de agradecimiento al creador de un nuevo contenido blog. Obviamente cuantos más hooks se demuestre que se conoce mejor se evaluará el ejercicio.