Programació de components MVC per Joomla! 1.5.x

CESI
Informàtica i Comunicacions

Pl. President Tarradellas, 11 ● **FIGUERES** (Girona)
Tel. 902 88 92 67 ● Fax 972 67 19 62

Autor: Carles Serrats

Pl. President Tarradellas, 11 ● **FIGUERES** (Girona)
Tel. 902 88 92 67 ● Fax 972 67 19 62

Índice

Pl. President Tarradellas, 11 ● **FIGUERES** (Girona)
Tel. 902 88 92 67 ● Fax 972 67 19 62

Pl. President Tarradellas, 11 ● **FIGUERES** (Girona)
Tel. 902 88 92 67 ● Fax 972 67 19 62

Preámbulo

Requisitos previos - comprensión sólida de cómo instalar y configurar Joomla y extensiones de terceros; comprensión básica de la administración de la página Web, incluyendo como instalar y configurar PHP, Apache, y MySQL y el modo de realizar copias de seguridad y restaurar una Joomla base de datos y página web Joomla; conocimiento práctico de XHTML y CSS, o bien algunos conocimientos de programación o voluntad para trabajar duro para aprender los conceptos básicos.

Conceptos Básicos

Los componentes son pequeños programas que se instalan en Joomla que permiten la manipulación de los datos hacia un objetivo particular.

En esta nueva versión se realizo una completa reestructuración del sistema con mejoras en todos los aspectos

Patrón de diseño MCV

La nueva versión 1.5 de joomla! Incluye novedades en la elaboración de componentes, entre ellas la posibilidad de incorporar un patrón de diseño MVC , que a la larga facilitará el mantenimiento de nuestro componente, y lo hará más fácil de extender al mismo tiempo que resultará más fácil de leer por otras personas.

Lo primero que hay que tener claro es el funcionamiento del modelo MVC. MVC son las siglas de Model View Controller, es decir, modelo vista controlador. Una aplicación Web basada en este patrón separar su código en tres partes diferenciadas:

- **El controlador:** el controlador es el punto de entrada de la aplicación, se mantiene a la escucha de todas las peticiones, ejecuta la lógica de la aplicación, y muestra la vista apropiada para cada caso.

- **El modelo:** el modelo contiene todo el código relacionado con el acceso a datos. Es importante que sea un código lo más genérico posible y se pueda reutilizar en otras situaciones y proyectos. Nunca incluiremos lógica en el modelo, solamente consultas a la base de datos y validaciones de entrada de datos.

- **La vista:** la vista contiene el código que representará lo que nosotros veremos por pantalla, en este caso se trata de código html.

El objetivo de usar este patrón de diseño, es separar separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos. El patrón MVC se ve frecuentemente en aplicaciones Web, donde la vista es la página HTML y el código que provee de datos dinámicos a la página, el modelo es el Sistema de Gestión de Base de Datos y el controlador representa la Lógica de negocio.

Pl. President Tarradellas, 11 ● **FIGUERES** (Girona)
Tel. 902 88 92 67 ● Fax 972 67 19 62

Implementación del MVC de Joomla! 1.5

El marco de trabajo de Joomla! proporciona una implementación sin archivos de configuración. Joomla no proporciona este archivo de configuración, sin embargo tiene otra forma de mapear las acciones del controlador. Joomla le da vital importancia al nombre que le tienes que dar a cada fichero del componente, es importantísimo que asignes bien el nombre del controlador, las vistas y los modelos de tu componente, de lo contrario Joomla no sabrá encontrarlos.

Introducción al concepto de front-end y back-end.

Nuestro componente tendrá dos partes diferenciadas: el front-end y el back-end.

> **Front-end :** el front-end es la parte del componente que es visible para el usuario de nuestro sitio Web. Se visualiza donde hayamos incluido la etiqueta <jdoc:include type=”component” /> en el template. Su código fuente se encuentra en el directorio components dentro del directorio principal de Joomla!.
>
> **Back-end :** es la parte del componente que se mostrará cuando entremos al sitio como administradores. El componente se mostrará donde lo hayamos especificado en el template del back-end. Su código fuente se encuentra en el directorio components dentro del directorio administrador dentro del directorio principal de Joomla!.-

Siguiendo el patrón MVC, vamos a crear un componente para joomla 1.5 al que llamaremos películas. El objetivo de esta primera toma de contacto es entender la metodología de trabajo para crear el componente de joomla! 1.5.

Back-end componente MVC con Joomla!

Iniciación. Estructura del directorio.

Lo primero que haremos será ir a la carpeta principal de la parte administrativa de Joomla en nuestro servidor Web. Dentro de esa carpeta localizar el directorio components *administrator/components*.

Observa que dentro del directorio existen otros directorios que empiezan por com_xxxx . Esta es la primera norma que pone Joomla!: los componentes se deben de ubicar en un directorio cuyo nombre empiece por com_ seguido del nombre del componente, por ejemplo para el componente películas seria com_peliculas.

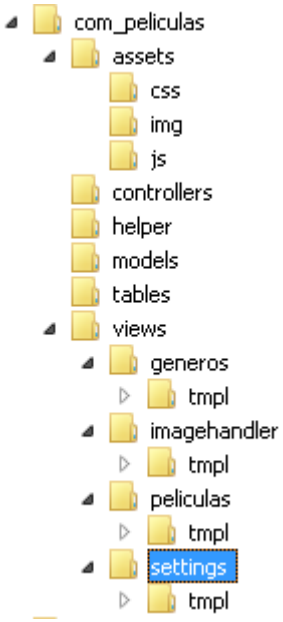
Cuando llamemos a nuestro componente, lo primero que hace Joomla es buscar el archivo php que hay dentro con el nombre “**admin.*nombre componente*.php**” y ejecutarlo. Este es el punto de entrada del componente. Creamos el fichero ***admin.peliculas.php***.

Por ahora solo estamos creando la estructura del componente. Lo siguiente es crear el archivo ***controller.php***, y de momento también lo dejaremos en blanco. Ahora

Pl. President Tarradellas, 11 ● **FIGUERES** (Girona)
Tel. 902 88 92 67 ● Fax 972 67 19 62

necesitamos crear los directorios en los que ubicaremos nuestros Modelos, Vistas y Controladores.

Creamos los siguientes directorios:



- **assets**. Directorio donde alojaremos los archivos auxiliares de css, js, imágenes, etc.…
  - **css :** Directorio para los css
  - **js :** Directorio para los archivos de javascript
  - **img :** Directorio para las imágenes
- **controllers**. Directorio donde alojaremos los archivos para cada controlador.
- **helper**. Directorio donde alojaremos archivos de clase para la ayuda en el desarrollo de nuestro componentes, como por ejemplo router.php
- **models**: Directorio donde alojaremos los archivos para cada modelo.
- **tables**: Directorio donde alojaremos los archivos de definición de las tablas de la base de datos
- **views**: Directorio donde alojaremos las subcarpetas para cada una de nuestras vistas. Para nuestro ejemplo crearemos las subcarpetas:
  - **generos**. Directorio para la vista de géneros.
    - **tmpl**. Directorio para los templates de la vista listado i formulario.
  - **imagehandler**. Directorio para la vista de la gestión de imágenes.
    - **tmpl**. Directorio para los templates de la vista listado i formulario.
  - **películas**. Directorio para la vista de las películas
    - **tmpl**. Directorio para los templates de la vista listado i formulario.
  - **settings**. Directorio para la vista de las configuraciones
    - **tmpl**. Directorio para los templates de la vista listado i formulario.

Cuando creamos las carpetas en Joomla!, debemos incluir una copia de index.html en cada uno. El archivo index.html en blanco es un archivo HTML, lo que impide que los usuarios puedan obtener de un listado de directorio.

Punto de entrada.

Cuando se carga el componente, se ejecuta el punto de entrada a este, ***admin.peliculas.php***, y en el punto de entrada será donde nosotros crearemos una instancia de nuestro componente.

Introduce el siguiente código en el punto de entrada de tu componente:

Pl. President Tarradellas, 11 ● **FIGUERES** (Girona)
Tel. 902 88 92 67 ● Fax 972 67 19 62

```php
<?php

// No permite el acceso directo
defined( '_JEXEC' ) or die( 'Restricted access' );

// Cargamos clases adicionales para la gestión de imágenes
require_once (JPATH_COMPONENT_SITE . DS . 'helpers' . DS . 'image.class.php' );
require_once (JPATH_COMPONENT_ADMINISTRATOR . DS . 'helper' . DS .
'peliculas.config.class.php' );

// Variable global de configuración del componente
global $Peliculas_Config;
$Peliculas_Config = new PeliculasConfig();
$Peliculas_Config->loadconfig();

// Añadimos el path del directorio de las tablas en el array de búsquedas del joomla
JTable::addIncludePath(JPATH_COMPONENT.DS.'tables');

// Cargamos el controlador base i la clase base de las vistas
require_once (JPATH_COMPONENT.DS.'base_controller.php');
require_once (JPATH_COMPONENT.DS.'base_view.php');

//Si viene en la petición, se importa el controlador específico
//El controlador específico hereda  del controlador base, por eso es necesario
//importar también el base
if( $controller = JRequest::getWord('controller') ) {
	$path = JPATH_COMPONENT.DS.'controllers'.DS.$controller.'.php';
	if (file_exists($path)) {
		require_once $path;
	} else {
		$controller = '';
	}
}

// Creamos nuestro propio controlador
$classname  = 'PeliculasController'.$controller;
$controller = new $classname( );

// Ejecutamos la tarea
$controller->execute( JRequest::getWord('task'));
$controller->redirect();
?>
```

Vamos a comentar el siguiente código paso a paso:

La primera línea comprueba si esta definida la variable “_JEXEC”, y si no esta definida se muestra un mensaje de error por pantalla mediante la función “die”. Esto es una medida de seguridad que incluye el marco de trabajo Joomla! y que es recomendable usar en todos nuestros archivos .php que tengamos en el sitio, y que evitara que la gente acceda a las paginas directamente sin hacer antes las comprobaciones de seguridad que lleva incluida Joomla!

Pl. President Tarradellas, 11 ● **FIGUERES** (Girona)
Tel. 902 88 92 67 ● Fax 972 67 19 62

Seguidamente se importan los archivos 'image.class.php' y 'peliculas.config.class.php' que contienen las configuraciones de nuestro controlador.

JPATH_COMPONENT y DS son constantes que define el marco de trabajo de Joomla! y que contienen el path al componente en el sistema y el separador de directorios adecuado para el sistema que se este utilizando, "\" para Windows y "/" para sistemas Unix. Utilizar estas constantes nos facilitara el trabajo más adelante y hará nuestras aplicaciones portables e independientes de la plataforma donde se estén utilizando.

Nombre	**Descripción**
DS	Separador de directorio (/)
JPATH_ADMINISTRATOR	Path del dir. de administración (/joomla/administrator)
JPATH_BASE	Path de entrada del directorio base (/joomla)
JPATH_CACHE	Path del Cache (/joomla/cache)
JPATH_COMPONENT	Path del Component (/joomla/components/com_peliculas)
JPATH_COMPONENT_ADMINISTRATOR	Path del directorio backend del Componente (/joomla/administrator/components/com_peliculas)
JPATH_COMPONENT_SITE	Path del directorio frontend del Componente (/joomla/components/com_peliculas)
JPATH_CONFIGURATION	Path de las configuraciones (/joomla)
JPATH_INSTALLATION	Path de la Instalación (/joomla/installation)
JPATH_LIBRARIES	Path de las Librerias (/joomla/libraries)
JPATH_PLUGINS	Path de los Plugins (/joomla/plugins)
JPATH_ROOT	Path del directorio de entrada del frontend (/joomla)
JPATH_SITE	Path del directorio público (/joomla)
JPATH_THEMES	Path del directorio de templates (/joomla/templates)

El archivo 'image.class.php' se encuentra en el font-end, dentro de la carpeta 'helpers' del componente, el archivo contiene las funciones para el tratamiento de las imágenes.
El archivo 'peliculas.config.class.php' se encuentra en la parte administrativa (Back-end) y contiene la clase que tiene las configuraciones de nuestro componente. Como vemos en el ejemplo automáticamente después de importar el archivo creamos una variable publica **$Peliculas_Config**, que será visible desde todo nuestro componente con la configuración de la aplicación.

Añadimos el path del directorio de las tablas en el array de búsquedas del joomla, de esta manera cuando tengamos que hacer referencia a una tabla, el joomla ya sabrá de donde la tiene que cargar.

Seguidamente se importa el fichero 'base_controller.php' donde crearemos nuestra clase base que contendrá el controlador. Todos los controladores que generemos a partir de ahora lo heredaremos de este archivo.

Pl. President Tarradellas, 11 ● **FIGUERES** (Girona)
Tel. 902 88 92 67 ● Fax 972 67 19 62

```php
<?php
// No permitimos el acceso directo
defined('_JEXEC') or die();

// Importamos el controlador de Joomla
jimport('joomla.application.component.controller');

// Iniciamos nuestro propio controlador 'PeliculasController'
class PeliculasController extends JController {

   function display() {
      parent::display();
   }
}
?>
```

El siguiente archivo 'base_view.php' lo utilizaremos para generar la clase base para todas las vistas que generemos a partir de ahora lo heredaremos de este archivo.

```php
<?php
// No permite el acceso directo
defined( '_JEXEC' ) or die( 'Restricted access' );

// Cargamos el componente base de vistas
jimport( 'joomla.application.component.view' );

class PeliculasView extends JView {

	function DisplayCss() {
		$document	= & JFactory::getDocument();
		$document->addStyleSheet(
'components/com_peliculas/assets/css/peliculasbackend.css' );
	}

	function Displayfooter() {
		echo	'<center><a	href="http://www.cesigrup.com"	target="_blank">CESI
Informatica i comunicacions</a></center>';
	}
}
?>
```

De esta manera siempre que tengamos una vista podremos lanzar la función "$this->Displayfooter()" y nos mostrara un pie de pagina uniforme para todas las vistas.

Después de importar el controlador, se comprueba si existe el parámetro *"controller"* en la query string, y si existe establece la variable path a JPATH_COMPONENT/controllers/$controller.php, para luego hacer un *require_once* de path, e importar el controlador que se especifica en la petición y que se sitúa en la carpeta *controllers* .

Pl. President Tarradellas, 11 ● **FIGUERES** (Girona)
Tel. 902 88 92 67 ● Fax 972 67 19 62

Después de importar el fichero correspondiente a nuestro controlador, lo instanciamos y ejecutamos el método *execute* del controlador, y le pasamos como parámetro un *string* que contiene el valor del parámetro *task* que hayamos establecido en la *query string*.

Después de esto se ejecuta el metodo *redirect()* del controlador, que redigirá el flujo del programa a la vista adecuada.

Géneros. 1er Contacto

Empezaremos con nuestro primer contacto en el modelo MVC.

El Controlador.

Lo primero que necesitamos es un controlador para nuestra vista géneros, creamos en la carpeta controllers, un archivo con nombre *generos.php* , dentro de este archivo crearemos la clase que contiene la lógica de negocio de la aplicación para el sistema de generos.

```php
<?php
/**
* @package PELICULAS
* @version 1.5.0
* @copyright Copyright (C) 2008 CESI Informatica i comunicions. All rights reserved.
* See COPYRIGHT.php for copyright notices and details.
*/

// No permite el acceso directo
defined( '_JEXEC' ) or die( 'Restricted access' );

// Importamos el controlador de Joomla
jimport('joomla.application.component.controller');

class PeliculasControllerGeneros extends PeliculasController
{
	/**
	 * Constructor
	 */
	function __construct()
	{
		parent::__construct();

		// Register Extra task
		$this->registerTask( 'add'  , 'edit' );
		$this->registerTask( 'apply', 'save' );
	}

	/**
	 * Lógica para guardar los datos
	 *
	 * @access public
	 * @return void
	 */
	function save()
	{
		// Comprueba del request la clave de falsificación
		JRequest::checkToken() or die( 'Invalid Token' );

		$task	= JRequest::getVar('task');

		// Recuperamos el post y lo saneamos si fuera necesario
		$post	= JRequest::get( 'post' );

		$model = $this->getModel('generos');

		if ($returnid = $model->store($post)) {
```

```
                                switch ($task)
                                {
                                        case 'apply' :
                                                $link =
'index.php?option=com_peliculas&view=generos&controller=generos&task=edit&cid[]='.$returnid;
                                                break;

                                        default :
                                                $link = 'index.php?option=com_peliculas&view=generos';
                                                break;
                                }
                                $msg = JText::_( 'GENERO GUARDADO' );

                                $cache = &JFactory::getCache('com_peliculas');
                                $cache->clean();

                        } else {

                                $msg    = '';
                                $link   = 'index.php?option=com_peliculas&view=generos';
                        }

                        $model->checkin();

                        $this->setRedirect($link, $msg);
                }

                /**
                 * Logica para publicar el genero
                 *
                 * @access public
                 * @return void
                 */
                function publish()
                {
                        $cid = JRequest::getVar( 'cid', array(0), 'post', 'array' );

                        if (!is_array( $cid ) || count( $cid ) < 1) {
                                JError::raiseError(500, JText::_( 'Seleccione un item a publicar' ) );
                        }

                        $model = $this->getModel('generos');

                        if(!$model->publish($cid, 1)) {
                                echo "<script> alert('".$model->getError()."'); window.history.go(-1);
</script>\n";
                        }

                        $total = count( $cid );
                        $msg = $total.' '.JText::_( 'GENERO PUBLICADO');

                        $this->setRedirect( 'index.php?option=com_peliculas&view=generos', $msg );
                }

                /**
                 * Logica para despublicar un Genero
                 *
                 * @access public
                 * @return void
                 */
                function unpublish()
                {
                        $cid = JRequest::getVar( 'cid', array(0), 'post', 'array' );

                        if (!is_array( $cid ) || count( $cid ) < 1) {
                            JError::raiseError(500, JText::_( 'Seleccione un item para despublicar' ) );
                        }

                        $model = $this->getModel('generos');

                        if(!$model->publish($cid, 0)) {
                           echo "<script> alert('".$model->getError()."'); window.history.go(-1);
</script>\n";
                        }
```

```
                $total = count( $cid );
                $msg     = $total.' '.JText::_( 'GENERO DESPUBLICADO');

                $this->setRedirect( 'index.php?option=com_peliculas&view=generos', $msg );
        }

        /**
         * Logic to orderup a category
         *
         * @access public
         * @return void
         */
        function orderup()
        {
                $model = $this->getModel('generos');
                $model->move(-1);

                $this->setRedirect( 'index.php?option=com_peliculas&view=generos');
        }

        /**
         * Logic to orderdown a category
         *
         * @access public
         * @return void
         */
        function orderdown()
        {
                $model = $this->getModel('generos');
                $model->move(1);

                $this->setRedirect( 'index.php?option=com_peliculas&view=generos');
        }

        /**
         * Logic to mass ordering categories
         *
         * @access public
         * @return void
         */
        function saveorder()
        {
                $cid = JRequest::getVar( 'cid', array(0), 'post', 'array' );
                $order = JRequest::getVar( 'order', array(0), 'post', 'array' );

                $model = $this->getModel('generos');
                $model->saveorder($cid, $order);

                $msg = 'New ordering saved';
                $this->setRedirect( 'index.php?option=com_peliculas&view=generos', $msg );
        }

        /**
         * Logic to delete categories
         *
         * @access public
         * @return void
         */
        function remove()
        {
                global $option;

                $cid = JRequest::getVar( 'cid', array(0), 'post', 'array' );

                if (!is_array( $cid ) || count( $cid ) < 1) {
                        JError::raiseError(500, JText::_( 'Select an item to delete' ) );
                }

                $model = $this->getModel('generos');

                $msg = $model->delete($cid);

                $cache = &JFactory::getCache('com_peliculas');
                $cache->clean();
```

```
                $this->setRedirect( 'index.php?option='. $option .'&view=generos', $msg );
        }

        /**
         * logic for cancel an action
         *
         * @access public
         * @return void
         */
        function cancel()
        {
                // Check for request forgeries
                JRequest::checkToken() or die( 'Invalid Token' );

                $category = & JTable::getInstance('peliculas_generos', '');
                $category->bind(JRequest::get('post'));
                $category->checkin();

                $this->setRedirect( 'index.php?option=com_peliculas&view=generos' );
        }

        /**
         * Logic to create the view for the edit categoryscreen
         *
         * @access public
         * @return void
         * @since 0.9
         */
        function edit( )
        {
                JRequest::setVar( 'view', 'generos' );
                JRequest::setVar( 'hidemainmenu', 1 );
                JRequest::setVar( 'layout', 'form' );

                $model    = $this->getModel('generos');
                $user     =& JFactory::getUser();

                // Error if checkedout by another administrator
                if ($model->isCheckedOut( $user->get('id') )) {
                        $this->setRedirect( 'index.php?option=com_peliculas&view=generos',
JText::_( 'EDITED BY ANOTHER ADMIN' ) );
                }

                $model->checkout();

                parent::display();
        }
}
?>
```

Como siempre la primera línea hace una comprobación para ver si se esta accediendo desde Joomla! o se esta haciendo un acceso directo, esta medida de seguridad debe de estar siempre presente.

La segunda línea se utiliza la función jimport del api de Joomla!, esta función importa la clase abstracta para el controllador.

Observa que la nomenclatura sigue la norma descrita:

Nombre del compoenente – Controller – Nombre del controlador

El controlador se mantiene a la escucha de todas las peticiones, se registran 2 tareas adicionales a las programadas por la clase base de joomla, la tarea “add” que ejecutara la función de “edit”, y la tarea “apply” que ejecutara la función “save”. Para las demás funciones estándar se ejecuta la lógica de la aplicación, para cada caso.

Pl. President Tarradellas, 11 ● **FIGUERES** (Girona)
Tel. 902 88 92 67 ● Fax 972 67 19 62

La función save() comprueba si la variable del formulario Token. Recoge los datos del post y carga el modelo géneros. A continuación llamamos la función del modelo "store" que nos almacenara los datos del post. El modelo nos devolverá la id donde ha guardado los datos y solo tenemos que comprobar si la tarea que teníamos que realizar era la de guardar o aplicar, para redireccionar a la vista correspondiente.

La función publish() y unpublish() realizan la acción de publicar y despublicar un elemento, la funcionalidad de las funciones es la misma. Se recoge el array de ID a publicar o despublicar, se carga el modelo y se pasa al modelo el array de ID para publicar y despublicar y se redirecciona a la vista.

La Vista.

Después de crear nuestro controlador, vamos a crear nuestra vista.
Vamos al fichero *view.html.php* en el directorio *views/generos.*

```php
<?php
/**
* @package PELICULAS
* @version 1.5.0
* @copyright Copyright (C) 2008 CESI Informatica i comunicions. All rights reserved.
* See COPYRIGHT.php for copyright notices and details.
*/

// No permite el acceso directo
defined( '_JEXEC' ) or die( 'Restricted access' );

// Importamos el controlador de Joomla
jimport( 'joomla.application.component.view');
```

Como siempre la primera línea hace una comprobación para ver si se esta accediendo desde Joomla! o se esta haciendo un acceso directo, esta medida de seguridad debe de estar siempre presente.

La segunda línea se utiliza la función jimport del api de Joomla!, esta función importa la clase abstracta para la vista.

```php
class PeliculasViewGeneros extends PeliculasView {
```

Observa la definición de la clase. Como se puede observar tiene un nombre un poco peculiar, establecido por el marco de trabajo Joomla!, y que consiste en poner primero el nombre del componente que lo llama, seguido de view y seguido del nombre de la vista:

NombreComponenteViewNombreVista

Esta norma es la que se sigue también para el modelo.

Pl. President Tarradellas, 11 ● **FIGUERES** (Girona)
Tel. 902 88 92 67 ● Fax 972 67 19 62

Lo que vamos a hacer en este caso es sobrescribir el método *display* y utilizar *assignRef* para crear dos “layout”. El del listado y el del formulario.

```
	function display($tpl = null)
	{
		global $mainframe, $option;

		//initialise variables
		$user		= & JFactory::getUser();
		$db		= & JFactory::getDBO();
		$document = & JFactory::getDocument();

		if($this->getLayout() == 'form') {
			$this->_displayForm($tpl);
			return;
		}
```

Lo primero que hacemos es comprovar el “layout”, si es “form” lo pasamos a la funcion que nos mostrara el formulario.

```
		JHTML::_('behavior.tooltip');

		//get vars
		$filter_order = $mainframe->getUserStateFromRequest(
$option.'.generos.filter_order', 'filter_order', 'g.ordering', 'cmd' );
		$filter_order_Dir = $mainframe->getUserStateFromRequest(
$option.'.generos.filter_order_Dir', 'filter_order_Dir', '', 'word' );
		$filter_state = $mainframe->getUserStateFromRequest(
$option.'.generos.filter_state', 'filter_state', '*', 'word' );
		$search   = $mainframe->getUserStateFromRequest( $option.'.generos.search',
'search', '', 'string' );
		$search   = $db->getEscaped( trim(JString::strtolower( $search ) ) );
```

Obtenemos los valores por defecto del filtrado de datos, asi como el orden y la dirección del orden. Para obtener estos datos utilizamos la classe global del framework principal del joomla “mainframe”, con la funcion “getUserStateFromRequest”, esta funcion obtiene el valor de la variable de estado (State) del usuario. El método se comprueba en primer lugar la variable especificada en el “request” para ver si un nuevo valor ha sido especificado. Si ha sido, entonces el valor de la variable de estado (State) se actualiza y el nuevo valor es devuelto. Si no fuera así, entonces se devuelve el valor almacenado se devuelve. Si no hay ningún valor especificado en el “request” y existe el valor en la variable “session”, entonces el valor por defecto es el devuelto.

```
		//create the toolbar
		JToolBarHelper::title( JText::_( 'GENEROS' ), 'generos' );
		JToolBarHelper::publishList();
		JToolBarHelper::spacer();
		JToolBarHelper::unpublishList();
		JToolBarHelper::spacer();
		JToolBarHelper::addNew();
		JToolBarHelper::spacer();
		JToolBarHelper::editList();
		JToolBarHelper::spacer();
		JToolBarHelper::deleteList();
		JToolBarHelper::spacer();
		JToolBarHelper::help( 'Peliculas.listgeneros', true );
```

Mediante este sistema, contruimos nuestro titulo y barra de acciones o toolbar.

Pl. President Tarradellas, 11 ● **FIGUERES** (Girona)
Tel. 902 88 92 67 ● Fax 972 67 19 62

```
		//Get data from the model
		$rows = & $this->get( 'Data');
		$pageNav  = & $this->get(  'Pagination'  );
```

Ahora consultaremos al modelo y obtendremos los datos a listar. Recordemos que Joomla busca un modelo con nombre *PeliculasModelGeneros* debido a que no se le pasa ningún parámetro a la función *getModel()* y lo instancia. Si le hubiéramos pasado el nombre de algún modelo como parámetro, nos hubiera devuelto el modelo especificado.

Una vez tenemos el modelo ejecutamos el método *getData* de este, que devuelve un *array* con los datos a listar. Tambien ejecutamos el método *getPagination* para obtener el objecto *pageNav* para la paginación. Ambas variables se le pasaran al layout por referencia mediante *assignRef*.

Pero antes contruimos una array en la variable lists para poner todos aquellos datos que necesitemos en la vista como por ejemplo, el texto a buscar, el orden, etc…

```
		//publish unpublished filter
		$lists['state']	= JHTML::_('grid.state', $filter_state );
		// search filter
		$lists['search']= $search;

		// table ordering
		$lists['order_Dir'] = $filter_order_Dir;
		$lists['order'] = $filter_order;

		$ordering = ($lists['order'] == 'g.ordering');
```

Y ahora si, passamos todos los datos al layout mediante *assignRef*.

```
		//assign data to template
		$this->assignRef('lists' , $lists);
		$this->assignRef('rows' , $rows);
		$this->assignRef('pageNav' , $pageNav);
		$this->assignRef('ordering' , $ordering);
		$this->assignRef('user' , $user);

		parent::display($tpl);
	}
```

Finalmente se ejecuta el método *display* de la clase padre, al cual se le puede pasar como parámetro el nombre del layout que se quiere visualizar, en nuestro caso no le pasaremos ninguno y visualizará el layout con nombre *default.php* .

Pero antes de empezar con el layout vamos a terminar con la vista para el caso en que tengamos que cargar el formulario de datos. Para ello contruimos la siguiente funcion.

```
	function _displayForm($tpl = null)
	{
		global $mainframe;

		//Load pane behavior
		jimport('joomla.html.pane');
```

```
// Initialize variables
$document = & JFactory::getDocument();
$user     = & JFactory::getUser();
$uri      = & JFactory::getURI();
$pane     = & JPane::getInstance('sliders');

$model    = & $this->getModel('generos');
$row      = & $this->get('Item');
```

Cargamos el modelo y lo consultamos con la funcion *getItem* para obtener el objecto con los datos que mostraremos, en nuestro caso la variable *$row* contine el genero a modificar.

```
//create the toolbar
if ( $row->id ) {
        JToolBarHelper::title( JText::_( 'EDIT GENERO' ), 'geenroedit' );
} else {
        JToolBarHelper::title( JText::_( 'ADD GENERO' ), 'generoedit' );
}
```

Empezamos a montar nuestra barra de tareas, y comprovamos el valor del $row-id, si tiene cualquier valor, quiere decir que el modelo nos ha cargado los datos y estamos modificando-los, si por el contrario no tiene valor o es 0 quiere decir que no hay dicho genero y estamos dando de alta uno nuevo.

```
JToolBarHelper::apply();
JToolBarHelper::spacer();
JToolBarHelper::save();
JToolBarHelper::spacer();
JToolBarHelper::media_manager();
JToolBarHelper::spacer();
JToolBarHelper::cancel();
```

Cargamos el resto de la barra de tareas.

```
// fail if checked out not by 'me'
if ($row->id) {
        if ($model->isCheckedOut( $user->get('id') )) {
                JError::raiseWarning( 'SOME_ERROR_CODE', $row->catname.'
'.JText::_( 'EDITED BY ANOTHER ADMIN' ));
                $mainframe->redirect(
'index.php?option=com_peliculas&view=generos' );
        }
}
```

Si estamos modificando el genero, vamos a consultar al modelo si podemos hacer-lo, mediante el metodo "*isCheckedOut*", este metodo, valida que nadie mas esta modificando este registro y lo bloquea para que los demas sepan que el usuario actual lo esta modificando.

```
// Get the lists
// build the html select list for ordering
$query = 'SELECT ordering AS value, nombre AS text'
. ' FROM #__peliculas_generos'
. ' ORDER BY ordering'
;
if($row->id) {
  $lists['ordering'] = JHTML::_('list.specificordering',  $row, $row->id, $query );
} else {
  $lists['ordering'] = JHTML::_('list.specificordering',  $row, '', $query );
}
```

Pl. President Tarradellas, 11 ● **FIGUERES** (Girona)
Tel. 902 88 92 67 ● Fax 972 67 19 62

```
                $lists['published'] = JHTML::_('select.booleanlist',  'published',
'class="inputbox"', $row->published );

                // Load the JEditor object
                $editor =& JFactory::getEditor();
```

Cargamos el resto de variables en el array de variables *$list*, que necesitamos para la modificación.
La funcion JHTML::_('list.specificordening') nos montará automáticamente un listbox con las ordenaciones según la consulta SQL que le passemos, pero esta consulta SQL debe de devolver 2 valores el "value" y el "text" para que monte bien el listbox.

La funcion JHTML::_('select.booleanlist') nos montará automáticamente un option group para la selección de Si/No.

```
                // Build the page title string
                $title = $row->id ? JText::_('Edit') : JText::_('New');

                // Set page title
                $document->setTitle($title);
```

Modificamos el titulo de la página.

```
                $this->assign('action',      $uri->toString());

                $this->assignRef('row',      $row);
                $this->assignRef('lists', $lists);
                $this->assignRef('user',     $user);
                $this->assignRef('pane',     $pane);

                parent::display($tpl);
        }
}
```

Passamos todos los datos al layout mediante *assignRef*.

Y finalmente se ejecuta el método *display* de la clase padre, al cual se le puede pasar como parámetro el nombre del layout que se quiere visualizar, en nuestro caso no le pasaremos ninguno y visualizará el layout con nombre *form.php* .

Layout.

Ahora vamos a ver los layouts a las que llama la vista. Un layout es la manera como se nos presentan los datos. La vista genera dos maneras de presentar los datos, la rimera default, que es un listado de los datos, en este caso es el listado de los generos. Y la ostra vista es el formulario que nos permitira añadir o modificar el genero.

Default.

Vamos al fichero *default.php* en el directorio *views/generos/tmpl*.

```
<?php

defined('_JEXEC') or die('Restricted access');
?>
```

Pl. President Tarradellas, 11 ● **FIGUERES** (Girona)
Tel. 902 88 92 67 ● Fax 972 67 19 62

Como siempre la primera línea hace una comprobación para ver si se esta accediendo desde Joomla! o se esta haciendo un acceso directo, esta medida de seguridad debe de estar siempre presente.

```
<form action="index.php" method="post" name="adminForm">
<table cellpadding="4" cellspacing="0" border="0" width="100%" class="adminlist">
<tr>
<td><img src="components/com_peliculas/assets/img/LogoCesi.png" height="97" width="233" alt="CESI
Informatica i comunicacions." align="left" /></td>
<td class="sectionname" align="right" width="100%"><font style="color: #C24733; font-size : 18px;
font-weight: bold; text-align: left;">::<?php echo JText::_( 'GENEROS' ); ?>::</font></td>
</tr>
</table>
```

Básicamente el listado esta dentro de un formulario que continene una tabla.

```
<table class="adminform">
<tr>
<td width="100%">
        <?php echo JText::_( 'SEARCH' ); ?>
        <input type="text" name="search" id="search" value="<?php echo $this->lists['search']; ?>"
class="text_area" onChange="document.adminForm.submit();" />
        <button onclick="this.form.submit();"><?php echo JText::_( 'Go' ); ?></button>
        <button onclick="this.form.getElementById('search').value='';this.form.submit();">
<?php echo JText::_( 'Reset' ); ?> </button>
        </td>
        <td nowrap="nowrap"><?php echo $this->lists['state']; ?></td>
</tr>
</table>
```

Mostramos una tabla de cabecera con el campo “search” para poder buscar y una lista que previamente hemos preparado en la vista, para filtro de datos.

```
<table class="adminlist" cellspacing="1">
<thead>
<tr>
        <th width="5"><?php echo JText::_( 'Num' ); ?></th>
        <th width="5"><input type="checkbox" name="toggle" value="" onClick="checkAll(<?php echo
count( $this->rows ); ?>);" /></th>
        <th class="title"><?php echo JHTML::_('grid.sort', 'GENERO', 'nombre', $this-
>lists['order_Dir'], $this->lists['order'] ); ?></th>
        <th width="1%" nowrap="nowrap"><?php echo JText::_( 'PUBLISHED' ); ?></th>
        <th width="80"><?php echo JHTML::_('grid.sort', 'REORDER', 'g.ordering', $this-
>lists['order_Dir'], $this->lists['order'] ); ?></th>
        <th width="1%"><?php echo JHTML::_('grid.order', $this->rows, 'filesave.png', 'saveordercat'
); ?></th>
        <th width="1%" nowrap="nowrap"><?php echo JHTML::_('grid.sort', 'ID', 'g.id', $this-
>lists['order_Dir'], $this->lists['order'] ); ?></th>
</tr>
</thead>
```

Preparamos la cabecera de las columnas a listar. Utilizaremos la funcion JHTML::_(‘gris.sort’ para que el propio joomla genere una cabecera que nos permitira ordenar los datos, y mantendra el orden ascendente y descendente.

```
<tfoot>
        <tr>
                <td colspan="7"><?php echo $this->pageNav->getListFooter(); ?></td>
        </tr>
</tfoot>
```

Pl. President Tarradellas, 11 ● **FIGUERES** (Girona)
Tel. 902 88 92 67 ● Fax 972 67 19 62

Para el pie del listado mostraremos el sistema de paginación proporcionado en la variable que nos facilita la vista.

```
<tbody>
	<?php
	$k = 0;
	for ($i=0, $n=count($this->rows); $i < $n; $i++) {
		$row = $this->rows[$i];
		$link =
'index.php?option=com_peliculas&amp;controller=generos&amp;task=edit&amp;cid[]='. $row->id;
		$published = JHTML::_('grid.published', $row, $i );
		$checked = JHTML::_('grid.checkedout', $row, $i );
	?>
	<tr class="<?php echo "row$k"; ?>">
		<td><?php echo $this->pageNav->getRowOffset( $i ); ?></td>
		<td width="7"><?php echo $checked; ?></td>
		<td align="left">
		<?php
		if ( $row->checked_out && ( $row->checked_out != $this->user->get('id') ) ) {
			echo htmlspecialchars($row->nombre, ENT_QUOTES, 'UTF-8');
			echo " (". htmlspecialchars($row->alias, ENT_QUOTES, 'UTF-8') . ")";
		} else {
			?>
			<span class="editlinktip hasTip" title="<?php echo JText::_( 'EDIT GENERO'
);?>::<?php echo $row->nombre; ?>">
			<a href="<?php echo $link; ?>">
			<?php echo htmlspecialchars($row->nombre, ENT_QUOTES, 'UTF-8'); ?>
	  <?php echo " (".htmlspecialchars($row->alias,ENT_QUOTES,'UTF-8').")"; ?>
			</a></span>
			<?php
		}
		?>
		</td>
		<td align="center">
			<?php echo $published; ?>
		</td>
		<td class="order" colspan="2">
			<span><?php echo $this->pageNav->orderUpIcon( $i, true, 'orderup', 'Move
Up', $this->ordering ); ?></span>

			<span><?php echo $this->pageNav->orderDownIcon( $i, $n, true, 'orderdown',
'Move Down', $this->ordering );?></span>

			<?php $disabled = $this->ordering ?  '' : '"disabled=disabled"'; ?>

			<input type="text" name="order[]" size="5" value="<?php echo $row-
>ordering; ?>" <?php echo $disabled; ?> class="text_area" style="text-align: center" />
		</td>
		<td align="center"><?php echo $row->id; ?></td>
	</tr>
	<?php $k = 1 - $k; } ?>
</tbody>
```

Para el cuerpo de la tabla preparamos la varibles para cada fila. Observemos que a la varible $link le passamos el controlador, la tarea "*task*" a realizar y el id de la lista. Mostramos cada columna, pero para la columna que contiene el link para editar primero comprovamos el "*checked_out*" para saber si podemos o no modificar el registro.

```
</table>
<p class="copyright">
	<?php echo $this->Displayfooter(); ?>
</p>

<input type="hidden" name="boxchecked" value="0" />
<input type="hidden" name="option" value="com_peliculas" />
<input type="hidden" name="controller" value="generos" />
<input type="hidden" name="view" value="generos" />
<input type="hidden" name="task" value="" />
```

Pl. President Tarradellas, 11 ● **FIGUERES** (Girona)
Tel. 902 88 92 67 ● Fax 972 67 19 62

```
<input type="hidden" name="filter_order" value="<?php echo $this->lists['order']; ?>" />
<input type="hidden" name="filter_order_Dir" value="" />
</form>
```

Acavamos la tabla y passamos las variables como hidden (ocultas) que necesitamos, el controlador, la vista, la tarea y los filtros para la ordenación.

Layout. Form.

Vamos al fichero *form.php* en el directorio *views/generos/tmpl.*

```
<?php
/**
* @package PELICULAS
* @version 1.5.0
* @copyright Copyright (C) 2008 CESI Informatica i comunicions. All rights reserved.
* See COPYRIGHT.php for copyright notices and details.
*/

defined('_JEXEC') or die('Restricted access'); ?>
```

Como siempre la primera línea hace una comprobación para ver si se esta accediendo desde Joomla! o se esta haciendo un acceso directo, esta medida de seguridad debe de estar siempre presente.

```
JHTML::_('behavior.tooltip');
jimport('joomla.utilities.date');

global $mainframe;
// Initialize variables
$db =& JFactory::getDBO();
$nullDate = $db->getNullDate();
?>
```

Cargamos las librerias de fechas y de tooltip.
Los tooltips son pequeños textos que aparecen cuando el usuario pasa el ratón sobre algunos elementos. Este método crea un tooltip que aparece al pasar sobre una imagen.

```
<script language="javascript" type="text/javascript">
function submitbutton(pressbutton) {
        var form = document.adminForm;
        if (pressbutton == 'cancel') {
                submitform( pressbutton );
                return;
        }

        // do field validation
        if (form.nombre.value == ""){
                alert( "<?php echo JText::_( 'ADD NAME GENERO' ); ?>" );
        } else {
                submitform( pressbutton );
        }
}
</script>
```

Escribimos un codigo javascript para la validación del formulario, recordar que aunque tengamos codigo javascript para mostrar un mensaje utilizamos la funcion JText::_(‘*mitexto*’).

Pl. President Tarradellas, 11 ● **FIGUERES** (Girona)
Tel. 902 88 92 67 ● Fax 972 67 19 62

Y empezamos con el formulario:

```
<form action="index.php" method="post" name="adminForm" id="adminForm">
<table cellspacing="0" cellpadding="0" border="0" width="100%">
<tr>
        <td><img src="components/com_peliculas/assets/img/LogoCesi.png" height="97"
width="233" alt="CESI Informatica i comunicacions." align="left" /></td>

        <td class="sectionname" align="right" width="100%"><font style="color:
#C24733; font-size : 18px; font-weight: bold; text-align: left;"><?php echo $this-
>row->id ? '::'.JText::_( 'EDIT CATEGORY' ).'::' : '::'.JText::_( 'ADD CATEGORY'
).'::';?></font>
        </td>
</tr>
</table>
```

Ponemos una pequeña cabecera que nos indicara cuando estamos modificando un registro y cuando estamos añadiendo un dato nuevo.

```
<table cellspacing="0" cellpadding="0" border="0" width="100%">
<tr>
    <td valign="top">
        <fieldset class="adminform">
        <legend><?php echo JText::_( 'Details' ); ?></legend>
    <table class="admintable">
        <tr>
            <td class="key">
                <label for="Nombre">
                    <?php echo JText::_( 'Nombre' ); ?>:
                </label>
            </td>
            <td>
                <input class="text_area" type="text" name="nombre" id="nombre"
value="<?php echo $this->row->nombre; ?>" size="50" maxlength="100" title="<?php echo
JText::_( 'TIPTITLEFIELD' ); ?>" />
            </td>
                </tr>
        <tr>
            <td nowrap="nowrap" class="key">
                <label for="alias">
                    <?php echo JText::_( 'Alias' ); ?>:
                </label>
            </td>
            <td colspan="2">
                <input class="text_area" type="text" name="alias" id="alias"
value="<?php echo $this->row->alias; ?>" size="50" maxlength="255" title="<?php echo
JText::_( 'TIPNAMEFIELD' ); ?>" />
            </td>
        </tr>
```

Como vemos para acceder al dato cargado utilizamos el **$this->row->**

```
        <tr>
            <td class="key">
                <?php echo JText::_( 'Published' ); ?>:
            </td>
            <td colspan="2">
                <?php echo $this->lists['published']; ?>
            </td>
```

Pl. President Tarradellas, 11 ● **FIGUERES** (Girona)
Tel. 902 88 92 67 ● Fax 972 67 19 62

```
        </tr>
        <tr>
            <td class="key">
                <label for="ordering">
                    <?php echo JText::_( 'Ordering' ); ?>:
                </label>
            </td>
            <td colspan="2">
                <?php echo $this->lists['ordering']; ?>
            </td>
        </tr>
    </table>
    </fieldset>
    </td>
```

Asi mismo utilizamos el mismo sistema para mostrar los datos que previamente hemos cargado en la variable lists : **$this->lists[‘***mivariable***’]**

```
<td valign="top" width="320" style="padding: 7px 0 0 5px">
<table width="100%" style="border: 1px dashed silver; padding: 5px; margin-bottom:
10px;">
        <?php
        if ( $row->id ) {
        ?>
        <tr><td>
                <strong><?php echo JText::_( 'Genero ID' ); ?>:</strong>
        </td><td>
                <?php echo $row->id; ?>
        </td></tr>
        <?php } ?>

        <tr><td>
                <strong><?php echo JText::_( 'Hits' ); ?></strong>
        </td><td>
                <?php echo $row->hits;?>
        </td></tr>

        <tr><td>
                <strong><?php echo JText::_( 'Created' ); ?></strong>
        </td><td>
                <?php
                if ( $row->created == $nullDate ) {
                        echo JText::_( 'New document' );
                } else {
                        echo JHTML::_('date',  $row->created,
JText::_('DATE_FORMAT_LC2') );
                }
                ?>
        </td></tr>

        <tr><td>
                <strong><?php echo JText::_( 'Modified' ); ?></strong>
        </td><td>
                <?php
                if ( $row->modified == $nullDate ) {
                        echo JText::_( 'Not modified' );
                } else {
                        echo JHTML::_('date',  $row->modified,
JText::_('DATE_FORMAT_LC2'));
                }
                ?>
        </td></tr>
```

```
</table>

        </td>
</tr>
</table>
```

Utilizamos una segunda tabla a la derecha para mostrar los datos del documento, el Id si es una modificación, la fecha de creación, y la fecha de modificación.
Nos fijamos que cuando trabajamos con fechas utilizamos la funcion **JHTML::_('date',** *mivariblefecha***,** *formato***)**

Formatos de fecha:
DATE_FORMAT_LC=%A, %d %B %Y
DATE_FORMAT_LC1=%A, %d %B %Y
DATE_FORMAT_LC2=%A, %d %B %Y %H:%M
DATE_FORMAT_LC3=%d %B %Y
DATE_FORMAT_LC4=%d.%m.%y
DATE_FORMAT_JS1=y-m-d

```
<?php echo JHTML::_( 'form.token' ); ?>
<input type="hidden" name="option" value="com_peliculas" />
<input type="hidden" name="id" value="<?php echo $this->row->id; ?>" />
<input type="hidden" name="controller" value="generos" />
<input type="hidden" name="view" value="generos" />
<input type="hidden" name="task" value="" />
</form>
<p class="copyright"><?php echo $this->Displayfooter();; ?></p>
 <?php
//keep session alive while editing
JHTML::_('behavior.keepalive');
?>
```

Para asegurar una seguridad del formulario utilizamos la funcion **JHTML::_('form.token')** que se debe utilizar en conjuncion con el **JRequest::checkToken()** en el controlador.

Ponemos las variables que necesitamos en hidden, el nombre del componente, el controlador, la vista, la tarea, y el ID.

La funcion JHTML::_('behavior.keepalive') añade las funciones javascript y el codigo necesario para las sesiones para la edición.

El Modelo.

El modelo contiene todo el código relacionado con el acceso a datos.

Recordamos las funciones que devemos implementar:

Vista general
getData() : Devuelve un array con todos los datos a listar
getPagination() : Devuelve el objecto de paginación.

Pl. President Tarradellas, 11 ● **FIGUERES** (Girona)
Tel. 902 88 92 67 ● Fax 972 67 19 62

Vista del formulario
getItem( ): Devulve el datos cargado en una variable.

Acciones generales desde el controlador
publish() : Publica o despublica el elemento.
move() : Canvia el orden de un unico elemento arriba o abajo
saveorder() : Guarda el orden de los ítems seleccionados
delete() : Elimina un registro.
checkout() : Bloquea un registro.
checkin() : Desbloquea un registro.
isCheckedOut() : Comprueba si el registro esta bloqueado.

Creamos al fichero *generos.php* en el directorio *models.*

```php
<?php
/**
* @package PELICULAS
* @version 1.5.0
* @copyright Copyright (C) 2008 CESI Informatica i comunicions. All rights reserved.
* See COPYRIGHT.php for copyright notices and details.
*/

// No permite el acceso directo
defined('_JEXEC') or die('Restricted access');

// Importamos el modelo de Joomla
jimport('joomla.application.component.model');
```

Como siempre la primera línea hace una comprobación para ver si se esta accediendo desde Joomla! o se esta haciendo un acceso directo, esta medida de seguridad debe de estar siempre presente.

La segunda línea se utiliza la función jimport del api de Joomla!, esta función importa la clase abstracta para el Modelo.

```php
class PeliculasModelGeneros extends Jmodel
{
        var $_data = null;
        var $_total = null;
        var $_pagination = null;
        var $_id = null;
 /**
  * Constructor
  */
function __construct()
{
        parent::__construct();

        global $mainframe, $option;

        $limit = $mainframe->getUserStateFromRequest( $option.'.limit', 'limit', $mainframe-
>getCfg('list_limit'), 'int');
        $limitstart = $mainframe->getUserStateFromRequest( $option.'.limitstart', 'limitstart', 0,
'int' );

        $this->setState('limit', $limit);
        $this->setState('limitstart', $limitstart);
```

```
        $array = JRequest::getVar('cid',  0, '', 'array');
        $this->setId((int)$array[0]);
}

function setId($id)
{
        $this->_id   = $id;
        $this->_data = null;
}
```

Observa que la nomenclatura sigue la norma descrita:

Nombre del compoenente – Model – Nombre del modelo

La función __construct() es la primera que se ejecuta al crear la clase. Utilizamos la funcion "getUserStateFromRequest" para recoger las variables del limites de pagina, y con la funcion "setState" la asignamos a la variable de session, asi podremos disponer de ella más adelante con la funcion "getState".

Recogemos el array "cid" y lo asignamos a la variable interna _id.

```
function getData()
{
        if (empty($this->_data))
        {
                $query = $this->_buildQuery();
                $this->_data = $this->_getList($query, $this->getState('limitstart'), $this-
>getState('limit'));
        }
        return $this->_data;
}

function getTotal()
{
        // Lets load the content if it doesn't already exist
        if (empty($this->_total))
        {
                $query = $this->_buildQuery();
                $this->_total = $this->_getListCount($query);
        }
        return $this->_total;
}

function getPagination()
{
        // Lets load the content if it doesn't already exist
        if (empty($this->_pagination))
        {
                jimport('joomla.html.pagination');
                $this->_pagination = new JPagination( $this->getTotal(), $this-
>getState('limitstart'), $this->getState('limit') );
        }
        return $this->_pagination;
}
```

La función "getData()" comprueba si la variable $datos esta vacia, y si lo esta, entonces almacena en ella los datos de la tabla en la variable.Para ello utiliza el metodo "getList()" . Este método es heredado por la clase padre y lo que hace es devolver una lista de objetos con los resultados de la consulta. Cada objeto de la lista tendrá tantas propiedades como campos tenga la tabla consultada.Por lo tanto en la variable $datos se almacena.

Pl. President Tarradellas, 11 ● **FIGUERES** (Girona)
Tel. 902 88 92 67 ● Fax 972 67 19 62

La función “getTotal()” comprueba si la variable $total esta vacia, y si lo esta, entonces almacena en ella el total de datos listados en la tabla. Para ello utiliza el metodo “getListCount()”. Este método es heredado por la clase padre. Y utilizamos la misma consulta Query que utilizamos en la funcion “getData()”.

La función “getPagination” comprueba si la variable $pagination esta vacia, y si lo esta, entonces devuelve el objecto JPagination, fijemonos que utliza la funcion “getTotal()” para saber el total de registros listados.

```
function _buildQuery()
{
        // Get the WHERE and ORDER BY clauses for the query
        $where = $this->_buildContentWhere();
        $orderby = $this->_buildContentOrderBy();

        $query = 'SELECT g.* '
                . ' FROM #__peliculas_generos AS g'
                . $where
                . $orderby
                ;

        return $query;
}

function _buildContentOrderBy()
{
        global $mainframe, $option;

        $filter_order = $mainframe->getUserStateFromRequest( $option.'.generos.filter_order',
        $filter_order_Dir = $mainframe->getUserStateFromRequest( $option.'.generos.filter_order_Dir',
'filter_order_Dir','', 'word' );

        $orderby = ' ORDER BY '.$filter_order.' '.$filter_order_Dir.', g.ordering';

        return $orderby;
}

function _buildContentWhere()
{
        global $mainframe, $option;

        $filter_state = $mainframe->getUserStateFromRequest( $option.'.generos.filter_state',
'filter_state', '', 'word' );
        $search  = $mainframe->getUserStateFromRequest( $option.'.generos.search', 'search', '',
'string' );
        $search  = $this->_db->getEscaped( trim(JString::strtolower( $search ) ) );

        $where = array();

        if ( $filter_state ) {
                if ( $filter_state == 'P' ) {
                        $where[] = 'g.published = 1';
                } else if ($filter_state == 'U' ) {
                        $where[] = 'g.published = 0';
                }
        }

        if ($search) {
                $where[] = ' LOWER(g.nombre) LIKE \'%'.$search.'%\' ';
        }

        $where = ( count( $where ) ? ' WHERE ' . implode( ' AND ', $where ) : '' );

        return $where;
}
```

Con la funcion “_buildQuery” contruimos la consulta Query para ello utilizaremos 2 funciones:

Pl. President Tarradellas, 11 ● **FIGUERES** (Girona)
Tel. 902 88 92 67 ● Fax 972 67 19 62

"buildContentWhere" contruye la parte del where de la consulta.
"buildContentOrderBy" construye la parte del order by.

```
function &getItem( )
{
	if ($this->_loadData())
	{
	} else {$this->_initData();}

	return $this->_data;
}

function _loadData()
{
	// Lets load the content if it doesn't already exist
	if (empty($this->_data))
	{
		$query = 'SELECT *'
			. ' FROM #__peliculas_generos'
			. ' WHERE id = '.$this->_id
			;
		$this->_db->setQuery($query);
		$this->_data = $this->_db->loadObject();
		return (boolean) $this->_data;
	}
	return true;
}

function _initData()
{
	$user = & JFactory::getUser();

	// Lets load the content if it doesn't already exist
	if (empty($this->_data))
	{
		$genero = new stdClass();
		$genero->id 			= 0;
		$genero->nombre 		= null;
		$genero->published 		= null;
		$genero->ordering 		= null;
		$genero->created_by 		= null;
		$genero->created 		= null;
		$genero->modified_by 		= null;
		$genero->modified 		= 0;
		$genero->author_id 		= $user->id;
		$genero->author_ip 		= null;
		$genero->checked_out 		= 0;
		$genero->checked_out_time 	= 0;
		$this->_data 			= $genero;
		return (boolean) $this->_data;
	}
	return true;
}
```

La funcion "getItem" devuelve un único registro de la variable interna _data, para ello, utilizamos la funcion "loadData" que ejecuta una consulta SQL para comprovar que exista el dato a cargar, la funcion "loadData" devuelve TRUE si lo ha encontrado y lo carga en la variable interna _data. Si el registro no existe utilizamos la funcion "initData" que nos devuelve un objecto inicializao.

```
function publish($cid = array(), $publish = 1)
{
	$user	=& JFactory::getUser();

	if (count( $cid ))
	{
		$cids = implode( ',', $cid );

		$query = 'UPDATE #__peliculas_generos'
```

```
                        . ' SET published = ' . (int) $publish
                        . ' WHERE id IN ('. $cids .')'
                        . ' AND ( checked_out = 0 OR ( checked_out = ' . (int) $user->get('id'). '
) )';
                $this->_db->setQuery( $query );
                if (!$this->_db->query()) {
                        $this->setError($this->_db->getErrorMsg());
                        return false;
                }
        }
        return true;
}
```

La funcion publish, modifical el registro (los registros) del array cid para publicar-lo o despublicar-lo.

```
function move($direction)
{
        $row =& JTable::getInstance('peliculas_generos', '');

        if (!$row->load( $this->_id ) ) {
                $this->setError($this->_db->getErrorMsg());
                return false;
        }

        if (!$row->move( $direction )) {
                $this->setError($this->_db->getErrorMsg());
                return false;
        }

        return true;
}
```

La funcion move, primero crea una instacia de tabla. Las definiciones de tablas las tendremos en el directorio *tables*. Asi podemos utilizar las funciones de load() para comprovar si el registro existe, y utilizar las funcion heredada de la instacia table para mover-lo en la direccion que devemos.

```
function saveorder($cid = array(), $order)
{
        $row =& JTable::getInstance('peliculas_generos', '');
        // update ordering values
        for( $i=0; $i < count($cid); $i++ )
        {
                $row->load( (int) $cid[$i] );

                if ($row->ordering != $order[$i])
                {
                        $row->ordering = $order[$i];
                        if (!$row->store()) {
                                $this->setError($this->_db->getErrorMsg());
                                return false;
                        }
                }
        }

        return true;
}
```

La funcion saveorder guarda el nuevo orden para todos los registros del array $cid, para ello instanciamos la tabla, y desde un bucle de todos id vamos ordenando cada uno de ellos.

Pl. President Tarradellas, 11 ● **FIGUERES** (Girona)
Tel. 902 88 92 67 ● Fax 972 67 19 62

```
function delete($cid)
{
        $cids = implode( ',', $cid );

        $query = 'DELETE FROM #__peliculas_generos'
                        . ' WHERE id IN ('. $cids .')';

        $this->_db->setQuery( $query );

        if(!$this->_db->query()) {
                $this->setError($this->_db->getErrorMsg());
                return false;
        }
}
```

La funcion delete, eliminara los registros de la tabla que esten en el array de $cids

```
function checkout($uid = null)
{
        if ($this->_id)
        {
                // Make sure we have a user id to checkout the group with
                if (is_null($uid)) {
                        $user    =& JFactory::getUser();
                        $uid     = $user->get('id');
                }
                // Lets get to it and checkout the thing...
                $item = & JTable::getInstance('peliculas_generos', '');
                return $item->checkout($uid, $this->_id);
        }
        return false;
}
```

Con la funcion “checkout” bloquearemos el registro que queremos modificar. Comprovamos que tenemos un id, comprovamos que tenemos el usuario (uid = UserId), si no tenemos el usuario lo cargamos de la clase maestra JFactory.
Instanciamos la tabla, y utilizamos la funcion checkout federada de la clase para bloquear el registro. Si ha sido satisfactorio, devuelve TRUE, sino devolvera FALSE.

```
function checkin()
{
        if ($this->_id)
        {       $item = & JTable::getInstance('peliculas_generos', '');
                return $item->checkin($this->_id);
        }
        return false;
}
```

Con la funcion “checkin” desbloquea el registro, instanciamos la tabla y utilizamos la funcion de la clase base de Jtable “checkin” para desbloquear el registro.

```
function isCheckedOut( $uid=0 )
{       if ($this->_loadData())
        {       if ($uid) {
                        return ($this->_data->checked_out && $this->_data->checked_out != $uid);
                } else {
                        return $this->_data->checked_out;
                }
        } elseif ($this->_id < 1) {
                return false;
        } else {
                JError::raiseWarning( 0, 'Unable to Load Data');
                return false;
        }
}
```

Pl. President Tarradellas, 11 ● **FIGUERES** (Girona)
Tel. 902 88 92 67 ● Fax 972 67 19 62

La funcion “isCheckedOut” comprueba si esta bloqueado el registro, para ello utilizamos la funcion que antes hemos creado “loadData” para cargar el registro. Una vez cargado comprovamos el campo checked_out.

```
function store($data)
{
	global $mainframe;

	jimport('joomla.utilities.date');

	$config = & JFactory::getConfig();
	$tzoffset = $config->getValue('config.offset');
	$user = & JFactory::getUser();

	$row  = & $this->getTable('peliculas_generos', '');

	// bind it to the table
	if (!$row->bind($data)) {
		JError::raiseError(500, $this->_db->getErrorMsg() );
		return false;
	}

	if (!$row->id) {
		$row->ordering = $row->getNextOrder();
	}

	// Are we saving from an item edit?
	if (!$row->id) {
		// Es nuevo
		$date = new JDate($row->created, $tzoffset);
		$row->modified = $nullDate;
		$row->modified_by = '';

		$row->created = $date->toMySQL();
		$row->created_by = $user->get('id');
	} else {
		// es Modificacion
		$date = new JDate($row->modified, $tzoffset);
		$row->modified = $date->toMySQL();
		$row->modified_by  = $user->get('id');
	}

	// Make sure the data is valid
	if (!$row->check()) {
		$this->setError($row->getError());
		return false;
	}

	// Store it in the db
	if (!$row->store()) {
		JError::raiseError(500, $this->_db->getErrorMsg() );
		return false;
	}

	return $row->id;
}


} // end of class Jmodel
?>
```

La funcion “store” guarda el registro. Importamos las funciones de fecha, y cargamos el usuario actual. Instanciamos la tabla de la clase Jtable. Asignamos los datos a la tabla, al instanciar la tabla por defecto esta vacia, por eso utilizamos la funcion “bind” que nos asigna los valores a cada campo.
Comprovamos el id, porque si es 0, quiere decir que es un registro nuevo, y los nuevos tienen como ordenación la ultima posición.

Pl. President Tarradellas, 11 ● **FIGUERES** (Girona)
Tel. 902 88 92 67 ● Fax 972 67 19 62

Comprovamos el id, porque si es 0, quiere decir que es un registro nuevo, los nuevos tienen la fecha de modificacion a null, y la fecha de creación a la fecha actual del sistema. Si ya existe el id quiere decir que es una modificación, asi que deveremos poner la fecha de modificación del sistema.
Ahora utilizamos las funciones de la clase JTable para validar los datos con la funcion "check" y si son correctos los guardamos con la funcion "store", de esta manera podemos devolver el ID que se le ha asignado.

La tabla.

Como hemos visto anteriormente en el modelo, utilizamos una instancia de tabla para la definición de campos, validación y trabajo con los datos de la tabla.

Creamos al fichero *peliculas_generos.php* en el directorio *tables.*

```php
<?php
// No permite el acceso directo
defined('_JEXEC') or die('Restricted access');

class peliculas_generos extends JTable
{
	var $id 			= null;
	var $nombre 		= null;
	var $alias 			= null;

	var $state 			= 0;
	var $hits 			= 0;
	var $meta_keywords 	= null;
	var $meta_description 	= null;
	var $published 		= null;
	var $created 		= null;
	var $created_by 	= null;
	var $modified 		= null;
	var $modified_by 	= null;
	var $checked_out 	= 0;
	var $checked_out_time 	= 0;
	var $ordering 		= null;

	/**
	* @param database A database connector object
	*/
	function peliculas_generos(& $db) {
		parent::__construct('#__peliculas_generos', 'id', $db);
	}

	// overloaded check function
	function check()
	{
		// Not typed in a category name?
		if (trim( $this->nombre ) == '') {
			$this->_error = JText::_( 'ADD NAME GENEROS' );
			JError::raiseWarning('SOME_ERROR_CODE', $this->_error );
			return false;
		}

		$alias = JFilterOutput::stringURLSafe($this->nombre);
```

```
                if(empty($this->alias) || $this->alias === $alias ) {
                        $this->alias = $alias;
                }

                /** check for existing name */
                $query = 'SELECT id FROM #__peliculas_generos WHERE nombre = '.$this-
>_db->Quote($this->nombre);
                $this->_db->setQuery($query);

                $xid = intval($this->_db->loadResult());
                if ($xid && $xid != intval($this->id)) {
                        JError::raiseWarning('SOME_ERROR_CODE', JText::sprintf('GENERO
NAME ALREADY EXIST', $this->nombre));
                        return false;
                }

                return true;
        }
}
?>
```

Las variables que pongamos en la clase son la estructura de la tabla. Siempre devemos poner las variables:

```
        var $alias              = null;

        var $published          = null;
        var $created            = null;
        var $created_by         = null;
        var $modified           = null;
        var $modified_by        = null;
        var $checked_out        = 0;
        var $checked_out_time   = 0;
        var $ordering           = null;
```

La función películas_generos debe de tener el mismo nombre que la clase, y crea la clase según la tabla “#__peliculas_generos”.

Con la función “check” comprobamos los datos antes de guardar-los. Comprobamos que tengamos un nombre. Asignamos un alias con la función “stringURLSafe” que nos convierte un nombre en una URL, ejemplo: Nombre: Películas de terror => películas-de-terror.

Y por último comprobamos que en la tabla no tengamos ningún otro registro con el mismo nombre.

Pl. President Tarradellas, 11 ● **FIGUERES** (Girona)
Tel. 902 88 92 67 ● Fax 972 67 19 62

Front-end componente MVC con Joomla!

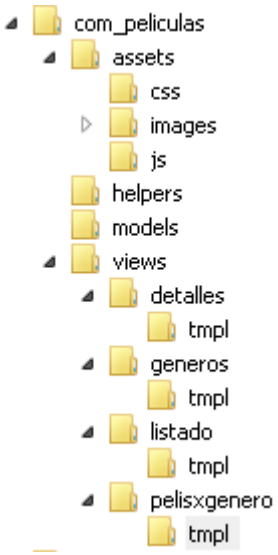
Iniciación. Estructura del directorio.

Lo primero que haremos será ir a la carpeta principal de Joomla en nuestro servidor web. Dentro de esa carpeta localizar el directorio components .
Observa que dentro del directorio existen otros directorios que empiezan por com_xxxx Esta es la primera norma que pone Joomla!: los componentes se deben de ubicar en un directorio cuyo nombre empiece por com_ seguido del nombre del componente, por ejemplo para el componente películas seria com_peliculas.

Cuando llamemos a nuestro componente, lo primero que hace Joomla es buscar el archivo php que hay dentro con el mismo nombre que el componente y ejecutarlo. Este es el punto de entrada del componente. Creamos el fichero peliculas.php.

Lo siguiente es crear el archivo controller.php.

Ahora necesitamos crear los directorios en los que ubicaremos nuestro Modelo (models) y nuestras Vistas (views).



Dentro del directorio views, hay que crear un directorio por cada vista y el directorio tiene que tener el nombre de la vista; crea el directorio *generos* dentro del directorio views.

Dentro del directorio views/generos creamos el directorio tmpl , aquí es donde se guardarán los layouts de la vista. El layout es el fichero final que se mostrará por el navegador.

Creamos el fichero *views/generos/view.html.php* , este fichero también es necesario.

Ahora creamos el fichero *generos.php* dentro de models.

Punto de entrada

Joomla! 1.5 sabe qué componente tiene que cargar porque busca en la *query string* de la petición el parámetro “option”, del cual obtiene el nombre del componente a cargar. Es decir, nosotros para llamar a nuestro componente *peliculas*, introduciremos en joomla la URL acabada con index.php?option=com_peliculas. Esto muestra la pagina principal de nuestro componente, que por defecto carga la vista con el mismo nombre del componente, es decir, la vista con nombre “peliculas”.

Cuando se carga el componente, se ejecuta el punto de entrada a este, peliculas.php, y en el punto de entrada será donde nosotros crearemos una instancia de nuestro componente.

Creamos al fichero *peliculas.php* en el directorio *com_peliculas.*

Pl. President Tarradellas, 11 ● **FIGUERES** (Girona)
Tel. 902 88 92 67 ● Fax 972 67 19 62

```php
<?php
// No permite el acceso directo
defined( '_JEXEC' ) or die( 'Restricted access' );

require_once (JPATH_COMPONENT_SITE.DS.'helpers'.DS.'image.class.php');
require_once (JPATH_COMPONENT_SITE.DS.'helpers'.DS.'output.class.php');
require_once (JPATH_COMPONENT_SITE.DS.'helpers'.DS.'route.php');
require_once
(JPATH_COMPONENT_ADMINISTRATOR.DS.'helper'.DS.'peliculas.config.class.php');

global $Peliculas_Config;
$Peliculas_Config = new PeliculasConfig();
$Peliculas_Config->loadconfig();

// Set the table directory
JTable::addIncludePath(JPATH_COMPONENT_ADMINISTRATOR.DS.'tables');

// Cargamos el controlador
require_once (JPATH_COMPONENT.DS.'controller.php');

// Create the controller
$classname  = 'PeliculasController';
$controller = new $classname();

// Perform the Request task
$controller->execute( JRequest::getVar('task', null, 'default', 'cmd') );

// Redirect if set by the controller
$controller->redirect();
?>
```

La primera línea comprueba si esta definida la variable "_JEXEC", y si no esta definida se muestra un mensaje de error por pantalla mediante la función "die". Esto es una medida de seguridad que incluye el marco de trabajo Joomla! y que es recomendable usar en todos nuestros archivos .php que tengamos en el sitio, y que evitara que la gente acceda a las paginas directamente sin hacer antes las comprobaciones de seguridad que lleva incluida Joomla!

Seguidamente se importan archivos, librerias de clase que utilizaremos para mostrar imágenes o rutas de acceso. Asi como tambien importamos desde la administración la classe base para las configuraciones, y inicializamos una variable pública con las configuraciones de nuestro componente.

Si deseamos trabajar con tablas importamos els path del directorio de las definiciones de las tablas.

Ahora importamos el archivo controller.php donde crearemos nuestra clase que contendrá el controlador.

JPATH_COMPONENT y DS son constantes que define el marco de trabajo de Joomla! y que contienen el path al componente en el sistema y el separador de directorios adecuado para el sistema que se este utilizando, "\" para Windows y "/" para sistemas Unix. Utilizar estas constantes nos facilitara el trabajo más adelante y hará nuestras aplicaciones portables e independientes de la plataforma donde se estén utilizando.

Instanciamos nuestro controlador, y ejecutamos el método execute del controlador, y le pasamos como parámetro un string que contiene el valor del parámetro task que hayamos establecido en la query string.

Después de esto se ejecuta el metodo redirect() del controlador, que redigirá el flujo del programa a la vista adecuada.

El controlador

Vamos al fichero *controller.php* y copiamos el siguiente codigo:

```php
 <?php
// No permite el acceso directo
defined( '_JEXEC' ) or die( 'Restricted access' );

jimport('joomla.application.component.controller');

class PeliculasController extends JController
{
        /**
         * Display the view
         */
        function display()
        {
                parent::display();
        }

}
?>
```

Esta vez lo único que hemos hecho es crear la clase que hará la función de controlador, y es **obligatorio que herede de *JController***.Hemos sobrescrito el metodo *display*, pero lo único que hacemos es ejecutar el metodo display de la clase padre, es decir, que básicamente no hacemos nada.

El controlador da paso a la vista correspondiente, que en este caso es la que se llame generos.

La vista

Después de crear nuestro controlador, vamos a crear nuestra vista.

Vamos al fichero *view.html.php* en el directorio *views/generos* y copiamos el siguiente código:

```php
 <?php
// No permite el acceso directo
defined( '_JEXEC' ) or die( 'Restricted access' );

jimport( 'joomla.application.component.view');

class PeliculasViewGeneros extends JView
{
        function display( $tpl=null )
```

Pl. President Tarradellas, 11 ● **FIGUERES** (Girona)
Tel. 902 88 92 67 ● Fax 972 67 19 62

```
        {
        global $mainframe;
        global $Peliculas_Config;

        $document = & JFactory::getDocument();
        $menu     = & JSite::getMenu();
        $item     = $menu->getActive();
        $params   = & $mainframe->getParams();

        $document->addStyleSheet(
JURI::base().'components/com_peliculas/assets/css/peliculas.css');

        $num_gen_page = $params->def('num_gen_page', 15);

        // get variables
        $limitstart = JRequest::getVar('limitstart', 0, '', 'int');
        $limit  = JRequest::getVar('limit', $num_gen_page, '', 'int');

        $rows   = & $this->get('Data');
        $total  = & $this->get('Total');

        $params->def( 'page_title', $item->name);

        //pathway
        $pathway = & $mainframe->getPathWay();
        $pathway->setItemName(1, $item->name);

        //Set Page title
        $mainframe->setPageTitle( $params->get('page_title') );
        $mainframe->addMetaTag( 'title' , $params->get('page_title') );

        // Create the pagination object
        jimport('joomla.html.pagination');
        $pageNav = new JPagination($total, $limitstart, $limit);

        $this->assignRef('rows' , $rows);
        $this->assignRef('params' , $params);
        $this->assignRef('pageNav' , $pageNav);
        $this->assignRef('item' , $item);
        $this->assignRef('Peliculas_Config' , $Peliculas_Config);
        $this->assignRef('pagetitle' ,  $pagetitle);

        parent::display($tpl);
        }
}
?>
```

Como siempre la primera línea hace una comprobación para ver si se esta accediendo desde Joomla! o se esta haciendo un acceso directo, esta medida de seguridad debe de estar siempre presente.

La segunda línea se utiliza la función jimport del api de Joomla!, esta función importa la clase abstracta para la vista.

Empezamos directamente con el metodo display. Definimos las variables globales mainframe y nuestra variable de configuraciones.

Obtenemos el objecto document, y el menú, concretamente el menú activo.

Pl. President Tarradellas, 11 ● **FIGUERES** (Girona)
Tel. 902 88 92 67 ● Fax 972 67 19 62

Mediante la variable document y la funcion addStyleSheet indicamos al joomla que en la cabecera del html (head) debe de cargar un css adicional para este componente.

Obtenemos el número de registros por página de la configuración de la vista. Esta configuración esta disponible en el XML del directorio *views/generos/tmpl/default.xml.*

Miramos si tenemos algun parámetro de los limites de paginación y si no lo tenemos lo inicializamos.

Ahora vamos a obtener los datos del modelo, no se lo hemos definido en ningun sitio, asi que el joomla cogera por defecto del objecto PeliculasModelGeneros. Podriamos haber definido el modelo al que acceder mediante $model = & $this->getmodel(); y obtener los datos por $model->get('Data'); pero asi ahorramos operaciones al componente, devido a que por defecto ya es lo que hace.

Obtenemos el titulo de la página de los parámetros del sistema (opción disponible cuando se define un enlace de menú desde la parte administrativa), para poder asignar al pathway el nombre que deseamos poner, en este caso el propio nombre de la página.

Ponemos el nombre de la página en el title del HTML mediante la función setPageTitle y también ponemos un Metatag con el mismo nombre con la función addMetaTag.

Importamos la librería de la clase pagination para obtener el objeto de la paginación.

Asignamos las variables al layout y lanzamos el metodo display que nos llamara al layout defautl del directorio tmpl.

Vista. Manifiesto (XML)

En el directorio de la vista vamos a crear el fichero ***metadata.xml.*** Este fichero nos dara las definiciones del listado para que lo podamos incorporar desde el apartado adminitrativo de menus. Son simplemente una descripción de lo que muestra el listado.

```
<?xml version="1.0" encoding="utf-8"?>
<metadata>
	<layout title="Listado de Generos">
		<message>
			<![CDATA[LISTADO DE GENEROS]]>
		</message>
	</layout>
	<state>
		<name>LISTADO DE GENEROS</name>
		<description>LISTADO DE GENEROS</description>
		<params>
			<param name="num_gen_page" type="text" size="3" default="15"
label="generos por pagina" description="PARAMGENPAGE" />
		</params>
	</state>
</metadata>
```

Pl. President Tarradellas, 11 ● **FIGUERES** (Girona)
Tel. 902 88 92 67 ● Fax 972 67 19 62

Tambien podremos definir alguna variable de configuración como por ejemplo "num_gen_page" que nos servira para decidir cuantos registros por pagina devemos mostrar.

El Layout.

En el directorio *views/generos/tmpl* vamos a crear el al fichero *default.php* y copiamos el siguiente código:

```
<?php
// No permite el acceso directo
defined( '_JEXEC' ) or die( 'Restricted access' );
?>

<div id="csiLista" style="margin: 0px;">
    <?php echo PELICULASOutput::header( JText::_( 'LISTADO DE GENEROS' ) ); ?>

        <table width='100%' border='0' cellspacing='1' cellpadding='4'
valign='middle'>
        <tfoot>
            <tr>
                <td><?php echo $this->pageNav->getPagesLinks(); ?></td>
            </tr>
        </tfoot>

        <tbody>
        <?php
        $linea=1;
        foreach ($this->rows as $row) :
            $linecolor = ($linea % 2) + 1;

            //Link to categoria details
            $detaillink = JRoute::_( 'index.php?view=pelisxgenero&id='. $row->slug );

            echo "<tr class='sectiontableentry".$linecolor."'><td width='100%'
valign='top'>";
            echo "<img
src='".JURI::base()."/components/com_peliculas/assets/images/film.png'
align='absmiddle' hspace='6' height='25' width='25' border='0' alt='".$this-
>escape($row->nombre)."'/></a>";
            echo "<a href='" . $detaillink . "'>".$this->escape($row->nombre)."</a>";
            echo " <span class='small'>(".$row->assignedpelis.")</span><br />";
            echo "</tr>";
            $linea++;
        endforeach;
        ?>
        </tbody>
        </table>
</div>

<!--pagination-->
<p class="pagescounter"><?php echo $this->pageNav->getPagesCounter(); ?></p>

<!--copyright-->
<p class="copyright"><?php echo PELICULASOutput::footer( ); ?></p>
```

El fichero no tiene mucho secreto. Ponemos una cabecera que tenemos definida en el fichero *helpers/output.class.php.*

Pl. President Tarradellas, 11 ● **FIGUERES** (Girona)
Tel. 902 88 92 67 ● Fax 972 67 19 62

Creamos una tabla donde incorporamos un bucle de todos los datos a listar, y los vamos poniendo en las celdas de la tabla, con su correspondiente link.

Layot. Manifiesto (XML)

En el directorio del layout vamos a crear el fichero ***default.xml.*** Este fichero nos dara las definiciones del layoutpara que lo podamos incorporar desde el apartado adminitrativo de menus. Son simplemente una descripción de lo que muestra el listado.

```
<?xml version="1.0" encoding="utf-8"?>
<metadata>
	<view title="Generos">
		<message>
			<![CDATA[LISTADO DE GENEROS]]>
		</message>
	</view>
</metadata>
```

Pl. President Tarradellas, 11 ● **FIGUERES** (Girona)
Tel. 902 88 92 67 ● Fax 972 67 19 62

Cosas

Creando URLs personalizadas

El Alias

El primer paso es la generación de lo que llamamos Alias. El Alias es usado en la URL en vez del título (el título es el texto que tu quieres tener en la URL). El alias debe ser "URI seguro", lo que quiere decir que los caracteres acentuados UTF-8 son substituidos por sus equivalentes en ASCII-7, los espacios en blanco por guiones, etcétera. El alias puede ser definido por el usuario, pero debes asegurarte que el requisito antes mencionado de la URL segura se cumple. Una buena manera de hacerlo es usuar el metodo JTable::check() durante el proceso de guardado.

```
function check()
{
	jimport( 'joomla.filter.output' );
	$alias = JOutputFilter::stringURLSafe( $this->title );
	if(empty( $this->alias ) || $this->alias === $alias ) {
		$this->alias = $alias;
	}
	/* All your other checks */
	return true;
}
```

El Slug

Continuando con el mismo ejemplo, el "slug" – "1-bienvenido-a-joomla" tiene dos partes. La primera parte es el identificador de artículo (id) y el segundo es el alias, estos están separados por un guión. Durante la consulta a la base de datos los dos elementos se combinan en el modelo:

```
$query = 'SELECT a.*'.
	' CASE WHEN CHAR_LENGTH (a.alias) THEN CONCAT_WS(\':\', a.id,
a.alias) ELSE a.id END as slug,'.
	[...];
```

Después de este paso "el slug" es usado en vez de la id..

Pl. President Tarradellas, 11 ● **FIGUERES** (Girona)
Tel. 902 88 92 67 ● Fax 972 67 19 62

Usar las Librerias de la Base de Datos 1.5

Conseguir el objeto de la base de datos del sistema

La manera más facil de conseguir el objeto de la base de datos es consiguiendo el objeto de la base de datos del sistema devuelto por la clase JFactory. El metodo statico JFactory::getDBO() devulve una instancia simple del objeto de la base de datos del sistema, sin importar el alcance y es la recomendación para conseguir este objeto.

```
// This gets the database object using the JFactory class
$db = &JFactory::getDBO();
```

Crear un objeto de base de datos a usar

Joomla! 1.5 introduce el concepto de manejadores de dase de datos que permitirán conseguir el objeto de las base de datos para diferentes fuentes de base de datos.

```
// Create a new database object
$myDB = new JDatabase::getInstance( $options );
```

Seleccione un campo de la fila.

Necesitamos fijar nuestra consulta SQL al objeto de la base de datos. Para hacer esto nosotros usaremos el metodo setQuery.

```
// Set our query to the database object.
$db->setQuery( 'SELECT nombre FROM tabla WHERE id = 22');
```

Finalmente llamamos al metodo loadResult().

```
// Use the loadResult() method to get only the first value from the fi
rst row of the result set.
$name = $db->loadResult();
```

La clase Error Handling

Vista general y Descripción:

En el Joomla 1.5 tenemos una nueva clase Error es JError class. Class está basado en patErrorManager que provee métodos estáticos para manejar la condición de error en Joomla. Previamente, no había un método consistente que maneje y muestre errores. Esta clase permite al desarrollador mostrar códigos de error y mensajes de error al usuario como también información interna para ayudar a solucionar el problema de fallas. Colocando un cierto código de error por una cierta condición de error, se puede entender mejor que sucede cuando un usuario se queja de cierta clase de error.

Pl. President Tarradellas, 11 ● **FIGUERES** (Girona)
Tel. 902 88 92 67 ● Fax 972 67 19 62

Implementación:

JError Class Static Methods:

isError(&$object):

Código PHP:

```php
/**
* method for checking whether the return value of a pat application method
is a pat
* error object.
*
* @static
* @access     public
* @param    mixed     &$object
* return    boolean $result           Verdadero si el argumento es un
patError-object, falso en caso contrario.
*/
function isError( &$object ) {
     return patErrorManager::isError($object);
}
```

Descripción: Error

raiseError($code, $msg, $info = null):

Código PHP

```php
/**
* envuelto por {@link raise()} método donde no tiene que especificar el
* nivel de error  – a {@link patError} objeto con nivel de error E_ERROR
será retornado.
*
* @static
* @access     public
* @param    string       $code    El código de error para esta aplicación-
interna.
* @param   string      $msg    El mensaje de error, también puede ser
mostrado al usuario si es necesario.
* @param    mixed       $info     Opcional: Información adicional al error
(usualmente información relevante para los desarrolladores que los
usuarios no ven, como una base de datos DSN).
* @return    object      $error    patError object configurado
* @see          patErrorManager
*/
function  &raiseError( $code, $msg, $info = null ){
     return patErrorManager::raise( E_ERROR, $code, $msg, $info );
}
```

Descripción: Warning

raiseWarning($code, $msg, $info = null):

Pl. President Tarradellas, 11 ● **FIGUERES** (Girona)
Tel. 902 88 92 67 ● Fax 972 67 19 62

Código PHP:

```php
/**
* envuelto por {@link raise()} método donde no tiene que especificar el
* nivel de error – a {@link patError} objeto con el nivel de error
E_WARNING será mostrado.
*
* @static
* @access      public
* @param     string     $code      El código de error de aplicación-
interna para este error
* @param     string     $msg       El mensaje de error, puede ser mostrado
al usuario si es necesario.
* @param      mixed    $ifo       Opcional: Información adicional al error
(usualmente información relevante para los desarrolladores que los
usuarios no ven, como una base de datos DSN).
* @return      object     $error     patError object configurado
* @see           patErrorManager
*/
function &raiseWarning( $code, $msg, $info = null ){
     return patErrorManager::raise( E_WARNING, $code, $msg, $info );
}
```

Descripción: Notice

raiseNotice($code, $msg, $info = null):

Código PHP

```php
/**
* wrapper for the {@link raise()} method where do you do not have to
specify the
* error level – a {@link patError} object with error level E_NOTICE will
be returned.
*
* @static
* @access      public
* @param     string     $code     The application-internal error code for
this error
* @param     string     $msg     The error message, which my also be shown
the user if need be.
* @param     mixed     $info      Optional: Additional error information
(usually only developer-relevant information that the user should never
see, like a database DSN).
* @return      object     $error     The configured patError object
* @see           patErrorManager
*/
function &raiseNotice( $code, $msg, $info = null ){
     return patErrorManager::raise( E_NOTICE, $code, $msg, $info );
}
```

Pl. President Tarradellas, 11 ● **FIGUERES** (Girona)
Tel. 902 88 92 67 ● Fax 972 67 19 62

Uso de los Desarrolladores:

Niveles de Error:

Error:

El nivel de error más serio es E_ERROR. Cuando aparece este tipo de error, el manejador del error parará la ejecución del script y mostrará el número del error y el mensaje en la pantalla. Ejemplos de cuando este error debe ser usado es cuando continúa una ejecución del script que puede causar la pérdida de la integridad de datos o un bucle infinito.

Warning:

El nivel intermedio de error es E_WARNING. Cuando este error aparece, el manejador del error mostrará un mensaje de advertencia con el número de error y el mensaje en pantalla permitirá al script continuar con la ejecución. Ejemplos de este caso pueden ser usado cuando se espera un dataset previsto no fue recuperado.

Nota:

El nivel de error más bajo es E_NOTICE. Este tipo de error se comporta de la misma manera que lo hace E_WARNING, pero implica un problema menos serio.

Consideraciones Especiales:

Porque la clase de JError envuelve solamente una cierta funcionalidad de la clase del patErrorManager, hay del patErrorManager que no se ponen en ejecución en JError. JError, sin embargo, se pone en ejecución de manera tal que permitiera que un desarrollador utilizara los plenos poderes del patErrorManager que juzgue necesarios.