Ez publish 3.* Tutorial
Introducción en el desarrollo de extensiones en Ez Publish

Dipl.-Ing. Felix Woldt (FH)
felix_[at]_jac-systeme.de
www.jac-systeme.de
Estado Abril 2007

Contenido

1 Introducción en el desarrollo de extensiones en eZ publish

1.1 Introducción

Muchos requisitos de un sistema de redacción se pueden cumplir con eZ Publish sin programación PHP.
Pero tarde o temprano cualquier autor llega a un punto, en el que un proyecto requiere requisitos específicos y hay que implementar extensiones (Extensions).

Este artículo deberá ayudar en los principios de esta materia mediante un simple ejemplo.

1.1.1 Que debería conocer/saber hacer...

_ Instalación de eZ publish y comprensión general de la estructura de eZ publish
_ Conocimientos generales en PHP, SQL, MYSQL, HTML son de utilidad.

1.1.2 Que se enseña

En éste articulo aprenderá como se crea una simple extensión y como se configura.
A la vez se muestra como puede ser utilizado el eZ Publish Framework en el desarrollo.

1.2 Que son extensiones

Antes de comenzar con el ejemplo, quería comentar el concepto de extension y entrar en el detalle de la estructuración de extensiones.

Una extension amplía la funcionalidad existente en eZ Publish, sin cambiar los datos originales. La separación de nuevas funcionalidades en extensiones tiene la ventaja, que serán posibles actualizaciones posteriores a una nueva version de eZ Publish.

Con extensiones es posible por ejemplo:

_ guardar el diseño de una pagina web / 'siteaccess' en una extension,
_ crear modulos propios con nuevos 'views' y plantillas 'fetch',
_ ampliar el sistema de plantillas con propios operadores de plantillas,
_ programar nuevos eventos Workflow, tipos de datos o Login Handler.

Esto solo es una pequeña seleccion de posibilidades.
Extensions en eZ publish a menudo tienen la misma estructura (ver Tabla 1).

Extension Subdirectorio	**Descripción**
actions	Nueva acción para formularios
autoloads	Definiciones de nuevos operadores de plantillas
datatypes	Definicion para nuevos tipos de data
design	Archivos (*.tpl, *.css, *.jpg, *.js ...) que tienen relación con el diseño
eventtypes	Propio definidos eventos Workflow
modules	Uno o mas modulos con funciones Views, Plantilla Fetch...
settings	Archivos de configuración (*.ini, *.ini.append.php) de la extension
translations	Archivos de traducción (*.ts)

Tabla 1: Posible estructura de directorio de una extension eZ publish 3.x

La estructura mostrada en la tabla 1 es solo un ejemplo y depende del tipo de la extension. No siempre son necesarios todas las carpetas, por ejemplo en una extension de operador de plantillas solo se necesita la carpeta autoloads y settings, en una extension de modulos se necesitan las carpetas modules y settings y quizas design.

1.3 Nueva extension mediante ejemplo de jacextension

El ejemplo de la extension jacextension deberá servir como entrada en los principios del desarrollo de extensiones eZ Publish

Se mostrarán todos los pasos necesarios para la creación de una extension y se presentarán las principales clases de PHP en eZ Publish Framework.
Se tratarán los siguientes casos:

_ Acceso a la base de datos (clase eZ Framework- eZPersistentObject)
_ Acceso a archivos de configuración .ini (eZ Framwork – clase eZINI)
_ Acceso a variables GET, POST SESSION (clase eZ Framework - eZHTTPTOOL)
_ Creación de propios mensajes Debug / Archivos Log (clase eZ Framework - eZDebug)
_ Uso del sistema de derechos para nuevos Views de un modulo
_ Ampliación del sistema de plantillas por propios funciones plantillas Fetch, es decir operadores

1.3.1 Requisitos

Para empezar con el ejemplo, eZ Publish deberá ser instalado correctamente.
Para ello establecemos una Plain Site mediante el asistente de eZ Publish,
en modo Access URL, en la base de datos MySql ez39_plain (juego de caraceres utf-8).
Asi se crean dos Siteaccess plain_site y plain_site_admin.

Las siguientes URL´s deberán funcionar posteriormente:

http://localhost/ez/index.php/plain_site (Vista de usuario)
http://localhost/ez/index.php/plain_site_admin (Vista de administrador)

asi localhost/ez/ remite directamente al directorio de instalacion de eZ Publish.

Implementar Extension

Queremos ahora crear una nueva extension con el nombre jacextension y activar ésta.
Ésta debe contener por el principio un modulo con el nombre modul1 con un View list que ejecuta el script PHP list.php.
Para ello entramos en la carpeta extension de la carpeta root en eZ Publish y creamos una nueva carpeta jacextension con los siguientes subdirectorios y archivos PHP:

Ezroot/ extension /
+--jacextension
+-- modules
+-- modul1
+-- list.php
+-- module.php
+--settings
+--module.ini.append.php

Con el archivo de configuración module.ini.append.php (Listing 1) comunicamos al sistema eZ Publish, que deberá buscar por modulos en la extension jacextension.
Por consiguiente eZ Publish intenta cargar todos los modulos como por ejemplo modul1, que se encuentran en el directorio extension/jacextension/modules/

Consejo: En entradas INI tener en cuenta que no se encuentren espacios libres al final de cada linea, ya que las variables no se interpretan correctamente.

1.3.2 View list

En module.php se definen todos los Views de un modulo (ver listing 2).
Mediante un view se puede acceder a un archivo PHP. En nuestro ejemplo definimos con $ViewList['list'] el nombre de la lista View y remitimos al archivo PHP list.php.

Queremos transmitir al View tambien unos parametros. Como ejemplar definimos en el archivo module.php dos parametros ParamOne y ParamTwo con 'params' => array('ParamOne', 'ParamTwo') .

La URL por consiguiente es:

http://localhost/ez/index.php/plain_site/list/ValueParamOne/ValueParamTwo/.

Posteriormente se puede acceder a las variables de list.php mediante el siguiente comando:

$valueParamOne = $Params['ParamOne']; und $valueParamTwo = $Params['ParamTwo'].

Esta presentacion de acceso de parametros se denomina ordered parameters. Ademas existen unordered parameters. Éstos se ajustan a los ordered parameters y se componen siempre de un Nombre-Valor-Pareja por ejemplo .. ...modul1/list/ValueParamOne/ValueParamTwo/ NombreParam3/ValueParam3/ NombreParam4/ValueParam4 .

La definición es parecida a la de los ordered parameters en module.php con 'unordered_params' => array('param3' => '3Param', 'param4' => '4Param').

Si el comando View por ejemplo tiene el valor .../parm4/141 en la URL entonces eZ Publish reconoce que el parametro param4 ha sido fijado y podemos leer el valor 141 del parametro Array del Modulo mediante $Params['4Param']

Los unordered parameters se pueden cambiar de orden. Si no se especifica un *unordered parameter* , entonces eZ Publish pone el valor a false.
En comparacion a esto, los *ordered parameters* deberán especificarse, en caso contrario reciben el valor NULL (cero).

Siguiente comando URL

http://localhost/ez/index.php/plain_site/modul1/list/table/5/param4/141/param3/131

especificaría los parametros view de la siguiente manera:

```
$Params['ParamOne'] = 'table';
$Params['ParamTwo'] = '5';
$Params['3Param'] = '131';
$Params['4Param'] = '141';
```

Ver tambien la documentación de eZ en
http://ez.no/doc/ez_publish/technical_manual/3_8/concepts_and_basics/modules_and_views .
Es usual crear un archivo PHP con el mismo nombre para cada View.
Asi se puede reconocer mediante un comando en la URL que archivo PHP se carga, por ejemplo:
http://localhost/ez/index.php/plain_site/content/view/full/2
remite al archivo view.php en el modulo Kernel content
(ezroot/kernel/content/view.php) .

Consejo: Se puede aprender mucho de los modulos Kernel de eZ, asi es conveniente mirar el codigo fuente de estos.
La estructura es identica a un modulo de una extension, como por ejemplo en nuestro ejemplo modul1.
Para poder visualizar la interpretación de la lista View en la pantalla, añadimos el comando echo en la list.php, la que interpreta los parametros en la pantalla (ver listing3).

1.3.3 Activación de la extension

Para comprobar la nueva lista View del modulo modul1 en la jacextension, debemos activar la extension.
Esto se realiza en la site.ini.append.php global (ver listing 4) o en la site.ini.append.php del Siteaccess correspondiente (ver listing 4).

A la vez la escritura se diferencia por ActiveExtensions[]
y ActiveAccessExtensions[].

La activación global es considerado para todos los Siteaccess de la instalación de eZ Publish. En nuestro ejemplo plain_site y plain_site_admin.

1.3.4 Sistema de derechos

Si intentamos ahora abrir la lista View del modulo modul1 con la URL
http://localhost/ez/index.php/plain_site/modul1/list/table/5/param4/141
Recibimos el mensaje de eZ Pulish “acceso denegado”. Porque? El usuario Anonymous no tiene los derechos necesarios para abrir la lista View del modulo modul1.

Para establecer los derechos necesarios hay dos posibilidades.
Primero podemos establecer un acceso para todos los usuarios mediante la entrada PolicyOmitList[]=modul1/list en el nuevo archivo de configuración extension/jacextension/settings/site.ini.append.php, (ver listing 6). O tambien podemos manejar el acceso mediante el sistema de Usuarios en eZ Publish. Para ello debemos cambiar el papel Anonymous para poder permitir el acceso a las funciones del modulo modul1. Esto se realiza mediante la URL:
http://localhost/ez/index.php/plain_site_admin/role/view/1.

Las funciones que puede tener un modulo de extension esta definido en module.php mediante el Array $FunctionList.
El vinculo se realiza mediante la definición en la View con las funciones Arraykey.
Asi vinculamos en nuestro ejemplo la lista View con la funcion de usuario read ($FunctionList['read'] = array()) con 'functions' => array('read').

Mediante $FunctionList es posible vincular Views individuales de un modulo con determinados funciones de usuario.

A la vez hay la función de usuario create, que está clasificado para todos los Views que crean contenidos.
Acceso a éstos solo podremos dar por ejemplo a redactores.

La funcion read vinculamos con todos los usuarios que deberán obtener derechos de lectura, asi por ejemplo tambien con el usuario Anonymous.

Listing 1. Archivo de configuración Modulo:

extension/jacextension/settings/module.ini.append.php

```
<?php /* #?ini charset="utf-8"?
# comunicar eZ, que busque por modulos en jacextension
[ModuleSettings]
ExtensionRepositories[]=jacextension
*/ ?>
```

Listing 2. Archivo de configuración View:

extension/jacextension/modul1/module.php

```
<?php
$Module = array( 'name' => 'Example Modul1' );
$ViewList = array();
// nueva lista View con 2 Parametros fijos y
// 2 parametros variables por orden
// http://.../modul1/list/ $Params['ParamOne'] / $Params['ParamTwo']/
param4/$Params['4Param'] /param3/$Params['3Param']
```

```
$ViewList['list'] = array( 'script' => 'list.php',
'functions' => array( 'read' ),
'params' => array('ParamOne', 'ParamTwo'),
'unordered_params' =>
array('param3' => '3Param', 'param4' => '4Param') );
// Las entradas en el papel de usuario
// se usan en la definición View, para que en el papel de usuario
// se puedan repartir derechos a propias funciones View
$FunctionList = array();
$FunctionList['read'] = array();
?>
```

Listing 3. Archivo de funcion de la lista View: extension/jacextension/modul1/list.php

```
<?php
// coger objeto actual del tipo eZModule
$Module =& $Params['Module'];
// leer parametro Ordered View
// http://.../modul1/list/ $Params['ParamOne'] / $Params['ParamTwo']
// por ejemplo .../modul1/list/view/5
$valueParamOne = $Params['ParamOne'];
$valueParamTwo = $Params['ParamTwo'];
// leer parametro UnOrdered View
//http://.../modul1/list/param4/$Params['4Param']/param3/$Params['3Param']
// por ejemplo .../modul1/list/.../.../param4/141/param3/131
$valueParam3 = $Params['3Param'];
$valueParam4 = $Params['4Param'];
// mostrar los valores de los parametros View
echo 'Example: modul1/list/'
.$valueParamOne .'/ '. $valueParamTwo.'/param4/'
.$valueParam4.'/ param3/'. $valueParam3;
?>
```

Listing 4. Posibilidad 1 – Activación de extensiones para todos los Siteaccess disponibles en el archivo de configruación global: settings/override/site.ini.append.php

```
<?php /* #?ini charset="utf-8"?
[ExtensionSettings]
ActiveExtensions[]
ActiveExtensions[]=jacextension
...
*/ ?>
```

Listing 5. Posibilidad 2 – Activación de extensiones en el archivo de configuración de un Siteaccess
Por ejemplo: settings/ siteacces/ plain_site/ site.ini.append.php

```
<?php /* #?ini charset="utf-8"?
...
[ExtensionSettings]
ActiveAccessExtensions[]=jacextension
...
*/ ?>
```

Listing 6. Poner los derechos de acceso a la lista View del modulo modul1 para todos los usuarios, en el archivo de configuración: extension/jacextension/settings/site.ini.append.php

```
<?php /* #?ini charset="utf-8"?
[RoleSettings]
PolicyOmitList[]=modul1/list
*/ ?>
```

1.3.5 Sistema de plantillas

Ya que el resultado del comando echo del script PHP list.php no nos satisface, queremos usar una plantilla propia. Para eso ponemos el archivo list.tpl en la carpeta jacextension/design/standard/templates/modul1/list.tpl.
Para que eZ Publish pueda encontrar despues la plantilla, tenemos que declarar la extension jacextension como extension de diseño.
Para eso creamos el archivo de configuración design.ini.append.php en la carpeta .../jacextension/settings/ (ver Listing 7).

En list.php creamos una instancia de plantilla TemplateInstance $tpl =& templateInit();

Luego ponemos el parametro View Array $viewParameters y el Array con los archivos de ejemplo $dataArray como variable de plantillas {$view_parameters}, es decir {$data_array} con la instrucción $tpl->setVariable('view_parameters', $viewParameters); y $tpl->setVariable('data_array', $dataArray);

Despues utilizamos encontrar/reemplazar para la plantilla list.tpl con las variables definidas.
En nuestro caso solamente $view_parameters y $dataArray y guardamos el resultado en $Result['content'].
En la plantilla principal de eZ Publish pagelayout.tpl se puede mostrar el resultado mediante la variable {$module_result.content}, lo que se hace por estandar.
Al final ponemos la ruta Modul1/list que ha de aparecer en el explorador (browser).

En nuestro ejemplo se puede hacer clic en la primera parte de la ruta, que enlaza a modul1/list (ver listing 8).
Ahora podremos acceder a las variables definidas en la plantilla list.tpl con {$view_parameters} y {$data_array}.
Los parametros View transmitidos mostramos mediante {$view_parameters|attribute(show)}
Despues comprobamos con el operador de plantillas (template operator) is_set($data_array) si la variable $data_array existe y emitimos una lista con los datos o un mensaje estándar (ver listing 9).
Si ahora abrimos nuestro View mediante http://localhost/ez/index.php/plain_site/modul1/list/table/5, no va a pasar mucho.
Solo aparece la ruta Modul1/list. Porque? No lo sabemos.
Para descubrir el fallo, activamos eZ Debug incluido las plantillas usadas en este momento. El compilar y cachear de las plantillas lo desactivamos para estar seguros de que se muestren todos los cambios de las plantillas.
Para eso expandimos la global ezroot/settings/override/site.ini.append.php con las siguientes entradas (ver listing 10).

Abrimos ahora.../modul1/list/table/5 de nuevo, deberia aparecer el mensaje de error
En la vista Debug:
'No template could be loaded for "modul1/list.tpl" using resource "design"' .
Supuestamente no se encuentra el archivo list.tpl.

En éste caso ayuda borrar el Cache de eZ Publish, ya que eZ ha cacheado la lista de las plantillas existentes.
Para eso abrimos la URL
http://localhost/ez/index.php/plain_site_admin/setup/cache
y le damos al boton “borrar todo el cache”. Ahora se debería mostrar la plantilla list.tpl con la vista de tablas, nuestros parametros view y nuestro ejemplo de lista de datos . Ademas deberá aparecer una entrada modul/list.tpl en la vista Debug 'Templates used to render the page:'
En nuestro ejemplo los parametros view tienen los siguientes valores $view_parameters. param_one = 'table ' y $view_parameters.param_one = '5' .

Éstos se pueden usar en el script PHP list.php o en la plantilla list.tpl, para controlar estados por ejemplo como debe aparecer lo emitido o que ID se usa.

Consejo: la variable de plantillas $view_parameters tambien está disponible en el Modul Kernel content de eZ, es decir en la mayoría de plantillas como por ejemplo node/view/full.tpl.

Listing 7. Declarar extension jacextension como extension de diseño

```
<?php /* #?ini charset="utf-8"?
# transmitir a eZ, a buscar por designs en jacextension
[ExtensionSettings]
DesignExtensions[]=jacextension
*/ ?>
```

Listing 8. modul1/list.php – Ampliación de Listing 3

```
<?php
// biblioteca para funciones de plantillas
include_once( "kernel/common/template.php" );
...
// inicializar Templateobject
$tpl =& templateInit();
// crear parametro View Array para poner en la plantilla
```

```
$viewParameters = array( 'param_one' => $valueParamOne,
'param_two' => $valueParamTwo,
'unordered_param3' => $valueParam3,
'unordered_param4' => $valueParam4);
// traspasar el parametro del View como Array a la plantilla
$tpl->setVariable( 'view_parameters', $viewParameters );
// crear Array de ejemplo en la plantilla => {$data_array}
$tpl->setVariable( 'data_array', $dataArray );
...
// usar buscar/reemplazar (parsen) en la plantilla y guardar el resultado para $module_result.content
$Result['content'] =& $tpl->fetch( 'design:modul1/list.tpl' );
...
?>
```

Listing 9. eZ Plantilla design:modul/list.tpl

```
{* list.tpl – Plantilla para Modulview .../modul1/list / ParamOne /
ParamTwo
Comprobar si existe la variable $data_array

- si: mostrar datos como lista
- no: mostrar mensaje
*}
Mostrar Array $view_parameters:
{$view_parameters|attribute(show)}<br />
<h1>Template: modul1/list.tpl</h1>
{if is_set($data_array)}
<ul>
{foreach $data_array as $index => $item}
<li>{$index}: {$item}</li>
{/foreach}
</ul>
{else}
<p>Atención: no existen datos!!</p>
{/if}
```

Listing 10. Activar vista Debug con lista de plantillas mediante archivo de configuración global:

ezroot/ settings/override/site.ini.append.php

```
<?php /* #?ini charset="utf-8"?
...
[DebugSettings]
Debug=inline
DebugOutput=enabled
DebugRedirection=disabled
[TemplateSettings]
ShowXHTMLCode=disabled
ShowUsedTemplates=enabled
TemplateCompile=disabled
TemplateCache=disabled
*/ ?>
```

1.3.6 View create

Ahora ampliamos nuestro ejemplo con un nuevo View create.
Queremos ahora guardar el Array con los datos de ejemplo en nuestra tabla de base de datos. Para eso creamos una nueva tabla de base de datos con el nombre jacextension_data en nuestra base de datos eZ ez39_plain con las columnas Id | user_id | created | value, por ejemplo con PhpMyAdmin (ver listing 11).

Listing 11. Ejecutar el siguiente comando SQL en nuestra base de datos eZ ez39_plain, para crear una nueva tabla jacextension_data.

```
CREATE TABLE `jacextension_data` (
`id` INT( 11 ) NOT NULL AUTO_INCREMENT PRIMARY KEY ,
`user_id` INT( 11 ) NOT NULL ,
`created` INT( 11 ) NOT NULL ,
`value` VARCHAR( 50 ) NOT NULL
) ENGINE = MYISAM ;
```

Después copiamos list.php hacía create.php, list.tpl hacia create.tpl y ampliamos module.php con un nuevo View y funcion de usuario create que enlaza hacía el archivo PHP create.php.
Luego cambiamos el llamamiento de plantillas en create.php a design:modul1/create.tpl y adaptamos los derechos para la View modul1/create. Borrar el cache posteriormente.

Ahora la llamada URL http://localhost/ez/index.php/plain_site/modul1/create debería funcionar (la misma vista que modul1/list solo sin parametros view).

Para poder guardar los nuevos datos en la base da datos MySql, necesitamos un formulario HTML que envía los datos a nuestra nueva View create (mediante variables POST o GET)

Para eso creamos un nuevo formulario en la plantilla create.tpl con una linea de texto name.
En nuestro ejemplo queremos enviar los datos mediante GET.
Asimismo insertamos una variable de plantillas {$status_message}
Para que podamos mostrar al usuario un mensaje (ver listing 12).

Listing 12. Plantilla jacextension/design/standard/templates/modul1/create.tpl con un formulario para guarder los nuevos datos

```
{* create.tpl – plantilla para la Modulview .../modul1/create
Formulario Html para guardar los nuevos datos *}
<form action={'modul1/create'|ezurl()} method="get">
Name:<br />
<input name="name" type="text" size="50" maxlength="50"><br />
<input type="submit" value="Crear nuevos datos">
<input type="reset" value="Cancelar">
</form>
<hr>
Status: {$status_message}
```

1.3.7 GET / POST

Luego cambiamos la list.php para que muestre la variable GET name, la que el formulario HTML transmite al script.

Despues mostramos ésta en el campo de estado de la plantilla. Para mostrar la variable GET / POST usamos el eZ Publish Framework con la clase eZHTTPTool. Primero cojemos la instancia del objeto de eZHTTPTool con $http =& eZHTTPTool::instance();
Mediante $http->hasVariable('name'); podemos descubrir si la variable $_GET['name'] o $_POST['name'] existen y mediante $http->variable('name'); lo podemos mostrar.

Si solo se quieren admitir variables GET o POST, se puede hacer mediante $http->hasGETVariable('name'); o $http->hasPOSTVariable('name');

Informacion acerca de funciones adicionales como por ejemplo acceder a Sessions, se puede leer en la documentación API de eZ Publish.

Para la clase eZHTTPTool se puede encontrar en la URL
http://pubsvn.ez.no/doxygen/classeZHTTPTool.html .

1.3.8 Debug

Después ponemos nuestra variable de plantillas {$status_message} con los valores del formulario o con un texto estándar. Mediante eZDebug::writeDebug()
se puede escribir el resultado o otro mensaje en la vista eZ Publish Debug y en el Archivo Log correspondiente por ejemplo ezroot/var/log/debug.log

Más instrucciones acerca la muestra de información son por ejemplo writeError() o writeNotice().

Se encuentra mas información en la documentación API en:
http://pubsvn.ez.no/doxygen/classeZDebug.html .

Si queremos escribir un archivo Log propio, lo podemos hacer mediante eZLog::write().
Ésto a veces es útil, ya que los archivos Log por defecto contienen información variada y puede ser poco claro de leer (ver Listing 13).

Listing 13. jacextension/module/modul1/create.php con ejemplos para mostrar variables GET/POST y generar mensajes Debug y archivos Log propios

```
<?php
// modul1/create.php – Archivo de función de View create
// anunciar biblioteca php necesaria para funciones de plantillas
include_once( "kernel/common/template.php" );
$Module =& $Params['Module'];
// cojer instancia de objeto global
$http =& eZHTTPTool::instance();
$value = '';
// Si la variable 'name' se emite por GET o  POST, mostrar variable
if( $http->hasVariable('name') )
$value = $http->variable('name');
if( $value != '' )
$statusMessage = 'Name: '. $value;
else
$statusMessage = 'Por favor introducir datos';
// inicializar Templateobject
$tpl =& templateInit();
$tpl->setVariable( 'status_message', $statusMessage );
// Escribir variable $statusMessage en el archivo eZ Debug Output / Log
// aqui los 4 tipos diferentes Notice, Debug, Warning, Error
eZDebug::writeNotice( $statusMessage, 'jacextension:modul1/list.php');
eZDebug::writeDebug( $statusMessage, 'jacextension:modul1/list.php');
eZDebug::writeWarning( $statusMessage, 'jacextension:modul1/list.php');
eZDebug::writeError( $statusMessage, 'jacextension:modul1/list.php');
// $statusMessage escribir propio archivo log hacía ezroot/
var/log/jacextension_modul1.log
```

```
include_once('lib/ezfile/classes/ezlog.php');
eZLog::write( $statusMessage, 'jacextension_modul1.log', 'var/log');
$Result = array();
// buscar/reemplazar plantilla y guardar resultado para $module_result.content
$Result['content'] =& $tpl->fetch( 'design:modul1/create.tpl' );
// generar ruta Modul1/create
$Result['path'] = array( array( 'url' => 'modul1/list','text' =>
'Modul1' ),
array( 'url' => false,'text' => 'create' ) );
?>
```

1.3.9 Acceso a base de datos

Queremos ahora volver a mencionar la base de datos. Queremos guardar el valor del formulario en nuestra nueva tabla jacextension_data.
Para esto en eZ Publish existe la clase eZPersistentObject. Ésta ya contiene las funciones para crear, cambiar, eliminar o extraer datos.
Para poder usar esta funcionalidad creamos una nueva clase JACExtenstionData
Ésta la guardamos en la carpeta ezroot/extension/jacextension/classes con el nombre jacextensiondata.php (ver Listing 14).

La función mas importante es JACExtenstionData::definition(). Aqui se define la estructura del objeto des JACExtensionData y se define en que tabla y con que columnas de tablas se guardarán los datos.

Despues creamos las funciones create(), fetchByID(), fetchList(), getListCount(). Se muestran tres diferentes maneras en que los datos se mostrarán (eZPersistentObject::fetchObject(), eZPersistentObject::fetchObjectList(), comando SQL directo).

Si es possible se deberían usar las funciones eZPersistentObject fetch. No se deberían usar entradas SQL específicos para base de datos, para que los comandos SQL puedan funcionar con el sistema de base de datos eZ Publish, por ejemplo Mysql, Postgres, Oracle.
(ver tambien API http://pubsvn.ez.no/doxygen/classeZPersistentObject.html)

Posteriomente usamos las funciones creadas en el script create.php. Para guardar nuevos datos, creamos un nuevo objeto del tipo JACExtensionData mediante la función JACExtensionData::create($value) . La función create() crea el value transmitido (valor del formulario) , la user_id actual y la hora actual.

Con la función store() guardamos ahora los datos en la tabla de base de datos jacextension_data. Para ver como han cambiado los datos, escribimos éstas en la vista Debug (ver Listing 15).

Para acceder a objetos del tipo eZPersitentObjekt, usamos $JacDataObject->attribute('id'). El parametro de funcion “id” corresponde a la columna de tablas id. Asi no tenemos que pensar mucho para acceder a los distintos valores.

De hecho esto vale para todos los datos que se guardan en eZ como por ejemplo objetos eZContentObject y eZUser.

Listing 14. jacextension/classes/jacextensiondata.php ejemplo para el acceso a la base de datos mediante eZ PersistentObjekt

```
<?php
include_once( 'kernel/classes/ezpersistentobject.php' );
include_once( 'kernel/classes/ezcontentobject.php' );
include_once( 'kernel/classes/datatypes/ezuser/ezuser.php' );
class JACExtensionData extends eZPersistentObject
{
/*!
Konstruktor
*/
function JACExtensionData( $row )
{
$this->eZPersistentObject( $row );
}
/*!
Definición de la estructura del objeto de datos /de la estructura de la tabla de base de datos
*/
function definition()
{
return array( 'fields' => array(
'id' => array( 'name' => 'ID',
'datatype' => 'integer',
'default' => 0,
```

```
'required' => true ),
'user_id' => array( 'name' => 'UserID',
'datatype' => 'integer',
'default' => 0,
'required' => true ),
'created' => array( 'name' => 'Created',
'datatype' => 'integer',
'default' => 0,
'required' => true ),
'value' => array( 'name' => 'Value',
'datatype' => 'string',
'default' => '',
'required' => true )
),
'keys'=> array( 'id' ),
'function_attributes' => array( 'user_object' => 'getUserObject' ),
'increment_key' => 'id',
'class_name' => 'JACExtensionData',
'name' => 'jacextension_data' );
}
/*!
Aqui ampliamos  attribute() del eZPersistentObject
*/
function &attribute( $attr )
{
if ( $attr == 'user_object' )
return $this->getUserObject();
else
return eZPersistentObject::attribute( $attr );
}
/*!
Función de ayuda se abre en la función attribute
*/
function &getUserObject( $asObject = true )
```

```
{
$userID = $this->attribute('user_id');
$user = eZUser::fetch($userID, $asObject);
return $user;
}
/*!
crea un nuevo objeto del tipo JACExtensionData y muestra éste
*/
function create( $user_id, $value )
{
$row = array( 'id' => null,
'user_id' => $user_id,
'value' => $value,
'created' => time() );
return new JACExtensionData( $row );
}
/*!
muestra los datos como JACExtensionData con id dada
*/
function &fetchByID( $id , $asObject = true)
{
$result = eZPersistentObject::fetchObject(
JACExtensionData::definition(),
null,
array( 'id' => $id ),
$asObject,
null,
null );
```

```
if ( is_object($result) )
return $result;
else
return false;
}
/*!
muestra todos los objetos JACExtensionData como objeto o como array
*/
function &fetchList( $asObject = true )
{
$result = eZPersistentObject::fetchObjectList(
JACExtensionData::definition(),
null,null,null,null,
$asObject,
false,null );
return $result;
}
/*!
muestra la cantidad de datos
*/
function getListCount()
{
$db =& eZDB::instance();
$query = 'SELECT COUNT(id) AS count FROM jacextension_data';
$rows = $db->arrayQuery( $query );
return $rows[0]['count']
}
// -- member variables--
//// \privatesection
var $ID;
var $UserID;
var $Created;
var $Value;
}
?>
```

Listing 15. jacextension/modules/modul1/create.php Creación de una nueva entrada en la base de datos y diferentes metodos para mostrarlos

```
<?php
// modul1/create.php – Archivo de función de View create
...
$value = '';
// Si variable 'name' se transmitió por GET o POST, mostrar variable
if( $http->hasVariable('name') )
$value = $http->variable('name');
if( $value != '' )
{
include_once('kernel/classes/datatypes/ezuser/ezuser.php');
// preguntar por la ID del usuario actual
$userId = eZUser::currentUserID();
include_once('extension/jacextension/classes/jacextensiondata.php');
// generar nuevo objeto de datos
$JacDataObject = JACExtensionData::create( $userId, $value );
eZDebug::writeDebug( '1.'.print_r( $JacDataObject, true ),
'JacDataObject antes de guardar: ID no definida) ;
// guarder objeto en base de datos
$JacDataObject->store();
eZDebug::writeDebug( '2.'.print_r( $JacDataObject, true ),
'JacDataObject después de guardar: ID definida') ;
// preguntar por la ID del nuevo objeto creado
$id = $JacDataObject->attribute('id');
// preguntar por el login del usuario que ha creado los datos
```

```
$userObject = $JacDataObject->attribute('user_object');
$userName = $userObject->attribute('login');
// mostrar de nuevo los datos
$dataObject = JACExtensionData::fetchByID($id);
eZDebug::writeDebug( '3.'.print_r( $dataObject, true ), 'JacDataObject
mostrado mediante función fetchByID()');
// investigar la cantidad de datos existentes
$count = JACExtensionData::getListCount();
$statusMessage = 'Nombre: >>'. $value .'<< del usuario >>'.$userName.'<<
En base de datos con id >>'.
$id.'<< guardado! Nueva cantidad = '.$count ;
}
else
{
$statusMessage = 'Por favor introducir datos';
}
// recoger datos como objeto y como array y mostrar en el Output Debug
$ObjectArray = JACExtensionData::fetchList(true);
eZDebug::writeDebug( '4. JacDataObjects: '.print_r( $ObjectArray, true ),
'fetchList( $asObject = true )');
$array = JACExtensionData::fetchList(false);
eZDebug::writeDebug( '5. JacDataArrays: '.print_r( $array, true ),
'fetchList( $asObject = false )');
// inicializar Templateobject
$tpl =& templateInit();
$tpl->setVariable( 'status_message', $statusMessage );
...

?>
```

1.3.10 Funcion plantilla Fetch (Template Fetch)

Sabemos ahora transmitir parametros y variables GET / POST a una view y mostrarlos. Pero si queremos mostrar datos por ejemplo de nuestra tabla de base de datos en una plantilla cualquiera de eZ Publish, no lo conseguiremos abriendo el View.
Para ésto hay las funciones Fetch que conocemos del modulo Kernel eZ por ejemplo . {fetch('content', 'node', hash('node_id', 2)}.
Queremos definir dos funciones Fetch, list y count, que dejan abrirse mediante el siguiente sintax de plantillas.
{fetch('modul1', 'list', hash('as_object' , true()))} und {fetch('modul1', 'count', hash())}.

La función list muestra todas las entradas de datos como Array o como objeto, y se define con el parametro propio definido 'as_object'.

La función count no tiene parametro y investiga mediante un simple comando SQL la cantidad de datos en nuestra tabla de base de datos jacextension_data.

La definición de las funciones Fetch se realiza en jacextension/modules/modul1/function_definition.php. Aqui se define que parametros se transmiten a que función PHP de una clase PHP determinada (ver Listing 16).

En el archivo jacextension/modules/modul1/ezmodul1functioncollection.php se encuentra una clase de ayuda, que contiene todas las funciones Fetch (ver Listing 17) .

Consejo: En ...functioncollection.php de cada modulo se pueden leer los posibles parametros para la parte hash() de una función Fetch.

Asi por ejemplo se puede investigar en el archivo kernel/content/ezcontentfunctioncollection.php que parametros hay para la funcion Fetch {fetch('content', 'tree', hash(...))} del modulo Kernel content de eZ.

Esto ayuda a menudo cuando la documentación de eZ Publish online está incompleta.

Listing 16. Definición de las funciones Fetch del modulo modul1 - extension/jacextension/modules/modul1/function_defintion.php

```
<?php
$FunctionList = array();
// {fetch('modul1','list', hash('as_object', true()))|attribute(show)}
$FunctionList['list'] = array(
'name' => 'list',
'operation_types' => array( 'read' ),
'call_method' =>
array('include_file' =>
'extension/jacextension/modules/modul1/ezmodul1functioncollection.php',
'class' => 'eZModul1FunctionCollection',
'method' => 'fetchJacExtensionDataList' ),
'parameter_type' => 'standard',
'parameters' => array( array( 'name' => 'as_object',
'type' => 'integer',
'required' => true ) )
);
//{fetch('modul1','count', hash())}
$FunctionList['count'] = array(
'name' => 'count',
'operation_types' => array( 'read' ),
'call_method' => array(
'include_file' =>
'extension/jacextension/modules/modul1/ezmodul1functioncollection.php',
'class' => 'eZModul1FunctionCollection',
'method' => 'fetchJacExtensionDataListCount' ),
'parameter_type' => 'standard',
'parameters' => array()
);
?>
```

Listing 17. Clase de ayuda con funciones PHP que se usan en la definición del template Fetch en function_defintion.php

– extension/jacextension/modules/
modul1/ezmodul1functioncollection.php

```
<?php
include_once('extension/jacextension/classes/jacextensiondata.php');
class eZModul1FunctionCollection {
function eZModul1FunctionCollection()
{}
// se abre mediante('modul1', 'list', hash('as_object', $bool ))
// fetch
function fetchJacExtensionDataList( $asObject )
{
return array( 'result' => JACExtensionData::fetchList($asObject) );
}
// se abre mediante('modul1', 'count', hash() ) fetch
function fetchJacExtensionDataListCount( )
{
return array( 'result' => JACExtensionData::getListCount() );
}
}
?>
```

1.3.11 Template Operator

Otra manera de acceder a funciones en una propia extensión es usar operadores de plantillas (template operators)

eZ Publish ya contiene una multitud de operadores de plantillas.
Seguidamente definimos un nuevo operador de plantillas con el nombre ($result_type) con un parametro result_type.
Mediante el parametro queremos controlar si se muestran todos los datos de nuestra tabla de base de datos o solo la cantidad. El comando de plantillas {jac('list')}
Muestra un array de datos y {jac('count')} la cantidad de los datos.
En nuestro ejemplo se usan las mismas funciones que en el Template Fetch fetch('modul1', 'list' , ...) und fetch('modul1', 'count', ...)

La definición de los operadores de plantillas disponibles en nuestra extensión jacextension se realiza en el archivo extension/jacextension/autoloads/ eztemplateautoload.php (ver Listing 18).

Lo que hace el operador de plantillas se define en una propia clase PHP. En nuestro caso en JACOperator del archivo extension/jacextension/autoloads/ jacoperator.php (ver Listing 19).

Para que eZ Publish sepa que nuestra extensión jacextension contiene operadores de plantillas, tenemos que comunicarlo en el archivo de configuración extension/jacextension/settings/site.ini.append.php eZ publish mediante ExtensionAutoloadPath[]=jacextension (ver Listing 20).

Para comprobar nuestras funciones Template Fetch fetch('modul1', 'list', hash('as_object', true())) y fetch('modul1', 'count', hash()) del operador de plantillas jac('list') o jac('count') ampliamos la plantilla list.tpl de la lista View (ver Listing 21)

Abriendo el View por ejemplo abriendo la URL http://localhost/ez/index.php/plain_site/modul1/list/tableblue/1234
Se deberían mostrar aparte del ejemplo Array $data_array, los datos de la tabla de base de datos jacextension_data dos veces, por una vez mediante los operadores de plantillas y por otra parte de las funciones Template Fetch .

Asi que hay varias posibilidades de acceder a las mismas funciones de la propia extension en una plantilla.

Listing 18. extension/jacextension/autoloads/eztemplateautoload.php

```
<?php
// Que operadores se cargarán automáticamente?
$eZTemplateOperatorArray = array();
// Operator: jacdata
$eZTemplateOperatorArray[] = array( 'script' =>
'extension/jacextension/autoloads/jacoperator.php',
'class' => 'JACOperator',
'operator_names' => array( 'jac' ) );
?>
```

Listing 19. extension/jacextension/autoloads/jacoperator.php

```
<?php
/*!
Operator: jac('list') und jac('count') <br>
Count: {jac('count')} <br>
Liste: {jac('list')|attribute(show)}
*/
class JACOperator{
var $Operators;
function JACOperator( $name = "jac" ){
$this->Operators = array( $name );
}
/*!
Returns the template operators.
*/
function &operatorList(){
return $this->Operators;
}
/*!
\return true to tell the template engine that the parameter list
exists per operator type.
*/
function namedParameterPerOperator()
{
return true;
}
/*!
See eZTemplateOperator::namedParameterList
*/
function namedParameterList()
{
return array( 'jac' => array( 'result_type' =>
array( 'type' => 'string',
```

```
'required' => true,
'default' => 'list' )) );
}
/*!
Dependiendo de los parametros que se transmitieron fetch objetos JACExtensionData
{jac('list)} o contar datos {jac('count')}
*/
function modify( &$tpl, &$operatorName, &$operatorParameters,
&$rootNamespace, &$currentNamespace, &$operatorValue,
&$namedParameters){
include_once('extension/jacextension/classes/jacextensiondata.php');
$result_type = $namedParameters['result_type'];
if( $result_type == 'list')
$operatorValue = JACExtensionData::fetchList(true);
elseif( $result_type == 'count')
$operatorValue = JACExtensionData::getListCount();
}
};
?>
```

Listing 20. extension/jacextension/settings/site.ini.append.php

```
<?php /* #?ini charset="utf-8"?
...
# buscar por operadores de plantillas en jaceextension
[TemplateSettings]
ExtensionAutoloadPath[]=jacextension
*/ ?>
```

Listing 21. Comprobación de las funciones Template Fetch propias definidas y del operador de plantllas - extension/jacextension/design/standard/templates/modul1/list.tpl

```
<h2>Template Operator: jac('count') und jac('list')</h2>
Count: {jac( 'count' )} <br />
```

```
Lista: {jac( 'list' )|attribute(show)}
<hr>
<h2>Funciones Template Fetch:
fetch('modul1','count', hash() <br />y<br />
fetch('modul1','list', hash( 'as_object', true()) )</h2>
Count: {fetch( 'modul1', 'count', hash() )} <br>
Lista: {fetch( 'modul1', 'list', hash( 'as_object', true()))|
attribute(show)}
```

1.3.12 Archivo INI

Para terminar queremos crear un propio archivo .ini extension/jacextension/settings/jacextension.ini. En ésta ponemos todos los valores que hemos introducido en plantillas o modulos y valores que puedan variar en diferentes instalaciones de eZ Publish por ejemplo si se deben mostrar mensajes Debug especiales.

La default .ini se puede sobreescribir con los archivos jacextension.ini.append.php, por ejemplo en el Siteaccess.

El Listing 22 es un ejemplo para un archivo .ini y Listing 23 muestra como se accede mediante PHP. Ampliamos list.php para comprobar el acceso.
Las demás funciones para el acceso a archivo INI se pueden consultar en API: http://pubsvn.ez.no/doxygen/classeZINI.html.

Imagen 1 muestra de nuevo la estructura de nuestro ejemplo jacextension.

Listing 22. Archivo de configuración de la extensión jacextension – extension/jacextension/settings/jacextension.ini

```
[JACExtensionSettings]
# Should Debug enabled / disabled
JacDebug=enabled
```

Listing 23. Acceso PHP a archivos INI jacextension.ini – extension/jacextension/modules/modul1/list.php

```
...
// leer variable JacDebug del bloque INI [JACExtensionSettings]
// del archivo INI jacextension.ini
include_once( "lib/ezutils/classes/ezini.php" );
$jacextensionINI =& eZINI::instance( 'jacextension.ini' );
$jacDebug = $jacextensionINI->variable('JACExtensionSettings','JacDebug');
// Si Debug está activado haz algo
if( $jacDebug === 'enabled' )
echo 'jacextension.ini: [JACExtensionSetting] JacDebug=enabled'; ...
```

1.3.13 Estructura de extension de ejemplo 'jacextension' & Sourcecode

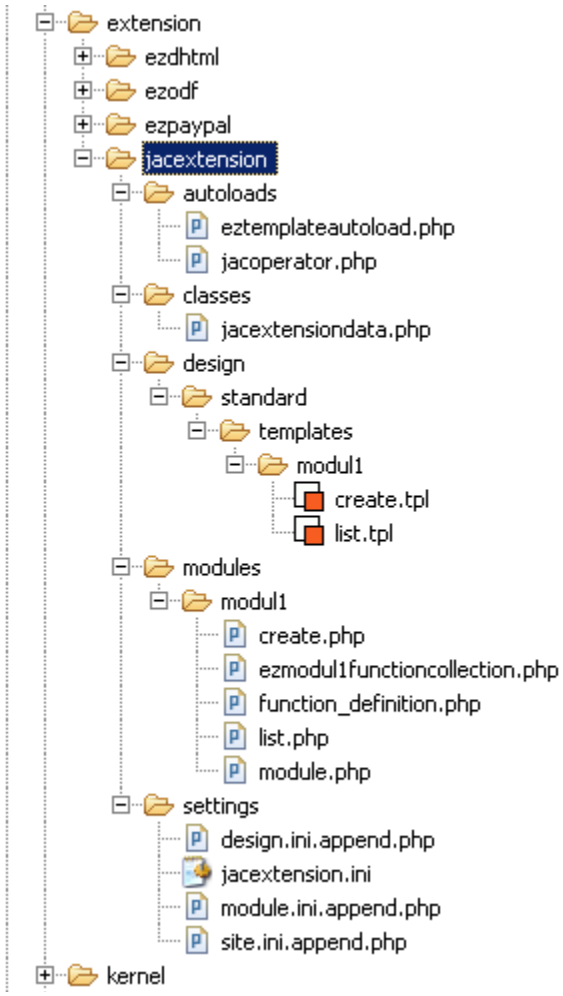

Imagen 1: Estructura de la extensión de ejemplo jacextension

El Sourcecode (codigo fuente) del Tutorial se puede descargar en internet en la dirección:
http://ez.no/community/contribs/examples/jacextension

1.4 Conclusión

Mediante un simple ejemplo de jacextension hemos aprendido las técnicas diferentes útiles para la creación de extensiones.
Aparte de crear propios modulos con diferentes Views y parametros Views, funciones Template Fetch, Operador de plantillas y el uso del sistema de derechos de eZ Publish sabemos ahora como escribir propios mensajes en la vista Debug o en archivos Log.

Se ha mencionado como se accede a archivos INI.

Con éste conocimiento básico debería ser posible ahora crear propias extensiones.

1.4.1 Enlaces em Internet

_ http://www.ezpublish.de/ – Comunidad Alemana eZ
_ http://www.ez.no/community - Comunidad International eZ
_ http://pubsvn.ez.no/doxygen/index.htm l – Documentación eZ API
_ http://ez.no/doc/ez_publish/technical_manual/3_8/reference – eZ documentación de referencia
_ http://ez.no/community/contribs/documentation/jac_dokumentation_in_german_ez_publish_basics_extension_development – PDF eZ publish principios en prorgamación de modulos (en alemán)

1.4.2 Sobre el Autor

Felix Woldt ha estudiado informática en la FH-Stralsund. Durante la carrera (desde 2002) y su diploma (2004) se ha relacionado con eZ publish y la programación de extensiones.
Felix es miembro activo de la comunidad alemana de eZ publish http://ezpublish.de y empleado en eZ Partner JAC Systeme http://www.jac-systeme.de