An Introduction to Developing eZ Publish Extensions

Felix Woldt

Monday 21 January 2008 12:05:00 am

Most Content Management System requirements can be fulfilled by eZ Publish without any custom PHP coding. But sooner or later experienced eZ Publish implementers get to the point where a project needs some special functionality and it becomes necessary to develop extensions.

This tutorial will help you with the basics of writing extensions using a simple example to illustrate the concepts.

Prerequisites

- An eZ Publish 3.x installation
- Basic understanding of the structure of eZ Publish
- Basic knowledge of PHP, SQL, MySQL and HTML

What will be demonstrated

In this tutorial you will learn how to create and configure a simple extension. It will also show how the eZ Publish framework can be used for development.

What are eZ Publish extensions?

Before talking about the example extension, I will explain the concept of extensions and talk about the structure of extensions.

An extension provides additional functionality (or extends existing functionality) in eZ Publish. It is called an extension because it plugs in to the eZ Publish core without requiring modifications to the original files. By placing new functionality in an extension, you can upgrade the eZ Publish version without affecting the extension.

Some examples of what you can do with extensions:

- save the design of a website / siteaccess
- create custom modules with new views and template fetch functions
- extend the template system with custom template operators
- program new workflow events, datatypes or login handlers

Extensions in eZ Publish usually adhere to a standard directory structure, as outlined in Table 1 below.

Extension subdirectory	**Description**
actions/	New actions for forms
autoloads/	Definitions of new template operators
datatypes/	Definitions for new datatypes
design/	Files (*.tpl, *.css, *.jpg, *.js ...) related to the design

eventtypes/	Custom workflow events
modules/	One or more modules with views, template fetch functions, and so on
settings/	Configuration files (*.ini, *.ini.append.php)
translations/	Translation files (*.ts)

Table 1: Standard directory structure of an eZ Publish 3.x extension

The directory structure shown above is only an example. The actual directory structure you use depends on the type of extension. Some directories may not be necessary. For example, a Template Operator extension only requires the directories *autoloads/* and *settings/*; in a module extension, only the directories *modules/* and *settings/*, and maybe a *design/* directory, are required.

Creating a new extension

Now we will study the example “jacextension” to learn the basics of extension development in eZ Publish. We will walk through the steps of creating an extension and will look at the basic PHP classes in the eZ Publish Framework.

The tutorial will cover the following concepts:

- Access to the database (eZ Framework – class eZPersistentObject)
- Access to .ini configuration files (eZ Framwork – class eZINI)
- Access to GET and POST session variables (eZ Framework – class eZHTTPTOOL)
- Creation of custom debug messages / log files (eZ Framework – class eZDebug)
- Use of the privilege system for new views of a module
- Extending the template system for your own template fetch functions or operators

Requirements

To work through the tutorial, eZ Publish 3.9 first must be properly installed. To do this, create a default site using the eZ Publish Setup Wizard, specifying “URL” in the siteaccess configuration and a MySQL database called “ez39_plain” (character set utf-8).

This creates two siteaccesses: plain_site and plain_site_admin, accessed through the following URLs:

http://localhost/ez/index.php/plain_site (User view)

http://localhost/ez/index.php/plain_site_admin (Administrator view)

(*localhost/ez/* is the root path of the eZ Publish directory.)

Setting up the extension

Now we will create and activate a new extension called “jacextension”. It will contain one module called “modul1”, with a view list that runs the PHP script *list.php*.

To do this, on the system command line we navigate to the directory *extension/* beneath the main eZ Publish directory and create a new directory called *jacextension/* with the following subdirectories and PHP files:

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

In the configuration file module.ini.append.php we tell eZ Publish to search for modules in jacextension. This will make eZ Publish try to load all modules (such as modul1), which are in the directory *extension/jacextension/modules/* .

Hint: In INI entries be careful not to leave blank spaces at the end of lines, as the variables might not be parsed correctly.

```
<?php /* #?ini charset="utf-8"?
# tell ez publish to search after modules in the extension jacextension
[ModuleSettings]
ExtensionRepositories[]=jacextension
*/ ?>
```

Listing 1. Module configuration file: extension/jacextension/settings/module.ini.append.php

Configuring and enabling the extension

View list

Module views are defined in the file *module.php*. A module view makes it possible to access a PHP file. In our example we define the name of the view list with $ViewList['list'] and forward to the custom PHP script *list.php*.

We also want to send some parameters to the view. For example, we define two parameters (ParamOne and ParamTwo) with the statement:

```
'params' => array('ParamOne','ParamTwo')
```

... in the script *module.php*.

```
<?php

// Introduction in the development of eZ Publish extensions

$Module = array( 'name' => 'Example Modul1' );
$ViewList = array();

// new View list with 2 fixed parameters and
// 2 parameters in order
// 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') );

// The entries in the user rights
// are used in the View definition, to assign rights to own View functions
// in the user roles

$FunctionList = array();
$FunctionList['read'] = array();

?>
```

Listing 2. View configuration file: extension/jacextension/modul1/module.php

The resulting URL is:

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

Later we can access the variables in *list.php* with the following commands:

```
$valueParamOne = $Params['ParamOne'];
$valueParamTwo = $Params['ParamTwo'];
```

This representation of parameter access is called ordered parameters. Unordered parameters adjust to the ordered parameters and consist of name-value pairs. For example:

```
...modul1/list/ValueParamOne/ValueParamTwo/ NameParam3/ValueParam3/NameParam4/ValueParam4
```

The definition is similar to the ordered parameters in module.php with:

```
'unordered_params' => array('param3' => '3Param', 'param4' => '4Param' )
```

For example, if the View command has the value .../param4/141, then eZ Publish recognises that the parameter "param4" has been specified and we can read value 141 from the parameter array of the module with the statement:

```
$Params['4Param']
```

Unordered parameters can change their order. If no unordered parameter is set, then eZ Publish sets the value to false. Ordered parameters must be defined; otherwise they are assigned a NULL (zero) value.

The URL http://localhost/ez/index.php/plain_site/modul1/list/table/5/param4/141/param3/131
specifies the parameter view as follows:

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

For more information, refer to the eZ Publish documentation at
http://ez.no/doc/ez_publish/technical_manual/3_9/concepts_and_basics/modules_and_views.

The general convention is to create a PHP file with a file name corresponding to the view name in the URL. This allows you to recognise from the URL which PHP file is being loaded. For example:

http://localhost/ez/index.php/plain_site/content/view/full/2

...is forwarding the *view.php* in the kernel module "content" (*ezroot/kernel/content/view.php*).

Hint: The kernel modules of eZ Publish are a valuable source of information and examples for programmers new to eZ Publish. The structure of kernel modules is the same as an extension, for example our modul1 example above.

To visualise the call of the view list on the screen, we add the "echo" command in *list.php*, which interprets the parameters on the screen. The listing below shows our code for *list.php*:

```
<?php

// take current object of type eZModule
$Module =& $Params['Module'];

// read parameter Ordered View
// http://.../modul1/list/ $Params['ParamOne'] / $Params['ParamTwo']
// for example .../modul1/list/view/5
$valueParamOne = $Params['ParamOne'];
$valueParamTwo = $Params['ParamTwo'];

// read parameter UnOrdered View
// http://.../modul1/list/param4/$Params['4Param']/param3/$Params['3Param']
// for example.../modul1/list/.../.../param4/141/param3/131
$valueParam3 = $Params['3Param'];
$valueParam4 = $Params['4Param'];

// show values of the View parameter
echo 'Example: modul1/list/'.
        $valueParamOne .'/ '.
        $valueParamTwo .'/param4/'.
        $valueParam4 .'/ param3/'.
        $valueParam3;

?>
```

Listing 3. Function file of view list: extension/jacextension/modul1/list.php

Activating the extension

To test the new view list of the module "modul1" in jacextension, we have to activate the extension. This is done in the global *site.ini.append.php* (see listing 4) or in the *site.ini.append.php* associated with the applicable siteaccess (see listing 5). In the global siteaccess, the extension is activated via the ActiveExtensions[] setting. If you are activating the extension for a specific siteaccess, use the ActiveAccessExtensions[] setting.

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

Listing 4. Option 1 – Activating the extension for all existing siteaccesses in the global configuration file: settings/override/site.ini.append.php

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

Listing 5. Option 2 – Activating extensions in the configuration file of a specific siteaccess, for example: settings/siteaccess/plain_site/site.ini.append.php

Privilege system

If we try to open the view list of modul1 with following URL:

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

...eZ Publish returns an “access denied” message. Why? Because the Anonymous user does not have the necessary privileges.

There are two ways to grant privileges. We could grant access to all users with the entry PolicyOmitList[]=modul1/list in the configuration file *extension/jacextension/settings/site.ini.append.php*, (see listing 6).

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

Listing 6. Set the access rights to the view list of modul1 for all users, in the configuration file extension/jacextension/settings/site.ini.append.php

Alternatively, we could alter the user access privileges using the eZ Publish Administration Interface. To do this we have to change the role Anonymous to allow access to the functions of module modul1. Use the following URL: http://localhost/ez/index.php/plain_site_admin/role/view/1.

The functions that can have an extension module are defined in *module.php* with the array $FunctionList. The link is done in the definition of the view with the Array key functions. So in our example we link the view list with the user function "read":

```
($FunctionList['read'] = array()) with 'functions' => array('read')
```

Using $FunctionList, it is possible to link individual views of a module with specific user functions. There is also the user function "create", which is assigned to all views that create content. In this case, we want to limit access to editors.

The "read" function is available to all the users who need read rights, and also for the Anonymous user.

Template system

Because the echo command in the PHP script *list.php* does not meet our needs, we want to use our own template. Therefore we put the file *list.tpl* into the folder *jacextension/design/standard/templates/modul1/*.

For eZ Publish to find the template, we have to declare jacextension as a design extension. To do this we create the configuration file *design.ini.append.php* in the folder *jacextension/settings/* (see Listing 7).

```
<?php /* #?ini charset="utf-8"?
# transmit to eZ, to search for designs in jacextension
[ExtensionSettings]
DesignExtensions[]=jacextension
*/ ?>
```

Listing 7. Declare jacextension as a design extension

In *list.php* we declare a variable $dataArray (array with strings). The values of this array we want to use later in the *list.tpl* template. To use the *list.tpl* template, first we have to initialize the template object:

```
$tpl =& templateInit();
```

Then we put the parameter View Array ($viewParameters) and the Array with the example files ($dataArray) as template variables {$view_parameters}, like {$data_array}, with the instruction:

```
$tpl->setVariable( 'view_parameters', $viewParameters );


$tpl->setVariable( 'data_array', $dataArray );
```

Next we do a find / replace for the template *list.tpl* with the defined variables (in our example only $view_parameters and $dataArray) and save the result in $Result['content'].

By default the main *pagelayout.tpl* template of eZ Publish shows the result with the variable {$module_result.content}. Finally we put Modul1/list in the navigation path (the "breadcrumbs") that are displayed in the browser. In our example this is done by clicking in the first part of the route, which links to modul1/list (see listing 8).

```
<?php
// library for template functions
include_once( "kernel/common/template.php" );

// Example array with strings
$dataArray = array <http://www.php.net/array>('Axel','Volker','Dirk','Jan','Felix');

...
// inicialize Templateobject
$tpl =& templateInit();
// create view array parameter to put in the template
$viewParameters = array(  'param_one' => $valueParamOne,
                                       'param_two' => $valueParamTwo,
                                       'unordered_param3' => $valueParam3,
                                       'unordered_param4' => $valueParam4 );
// transport the View parameter Array to the template
$tpl->setVariable( 'view_parameters', $viewParameters );

// create example Array in the template => {$data_array}
$tpl->setVariable( 'data_array', $dataArray );
...
// use find/replace (parsing) in the template and save the result for $module_result.content
$Result ['content'] =& $tpl->fetch ( 'design:modul1/list.tpl' );
...
?>
```

Listing 8. modul1/list.php – extending Listing 3

Now we have access to the defined variables in the template *list.tpl* with {$view_parameters} and {$data_array}. We show the transmitted view parameters with {$view_parameters|attribute(show)}. Next we use the template operator is_set($data_array) to see if the variable $data_array exists and send a list with the data or a default message (see listing 9).

```
{* list.tpl - Template for Modulview .../modul1/list/ParamOne/ParamTwo
Check if the variable $data_array exists
- yes: show data as list
- no: show message
*}

{*Show 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>Attention: no existing data!!</p>
{/if}
```

Listing 9. eZ Template design modul/list.tpl

If we now open our view with http://localhost/ez/index.php/plain_site/modul1/list/table/5, nothing much will happen. It only appears in the route Modul1/list. Why? We don't know yet.

To investigate the error, we activate eZ Debug, including the templates currently in use. We deactivate compiling and caching of templates to be sure that all changes in the templates will be shown. To do this we extend the global *ezroot/settings/override/site.ini.append.php* with the following entries (see listing 10).

```
<?php /* #?ini charset="utf-8"?

[DebugSettings]
Debug=inline
DebugOutput=enabled
DebugRedirection=disabled

[TemplateSettings]
ShowXHTMLCode=disabled
ShowUsedTemplates=enabled
TemplateCompile=disabled
TemplateCache=disabled

*/?>
```

Listing 10. Activate Debug view with template list over global configuration file: ezroot/settings/override/site.ini.append.php

We now open http://localhost/ez/index.php/plain_site/modul1/list/table/5 again. It generates an error message in the Debug view: 'No template could be loaded for "modul1/list.tpl" using resource "design"'.

It seems that the file *list.tpl* was not found. In this case it is useful to clear the cache of eZ Publish, as eZ has cached the list of the existing templates. So we load the URL http://localhost/ez/index.php/plain_site_admin/setup/cache and we click on “empty all caches”. Now the template *list.tpl* should appear with the table list, our view parameters and our example datalist . As well it should appear as *modul/list.tpl* in the Debug view “Templates used to render the page”.

In our example the viewparameters has the following values: $view_parameters.param_one='table' and $view_parameters.param_two='5'. The values of the viewparameters can be easily used to perform actions in PHP script *list.php* or in the template *list.tpl,* for example to display the ID or fetch some extended information to the given ID.

Hint: the template variable $view_parameters is also available in the kernel module content of eZ and therefore in most templates, such as *node/view/full.tpl*.

Creating a view

Now we extend our example by creating a new view. We want to save the array with the example data in the database, so we create a new database table (using, for example, phpMyAdmin) with the name "jacextension_data" in our ez39_plain database, with the columns id | user_id | created | value (see listing 11).

```
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 ;
```

Listing 11. Run the SQL command on the ez39_plaindatabase to create a new jacextension_data table.

Next we copy *list.php* to *create.php*, *list.tpl* to *create.tpl* and we extend *module.php* with a new view and user role function ("create"), which links to the PHP file *create.php*. Then we change the template call in *create.php* to *design:modul1/create.tpl* and adjust the rights for the view *modul1/create*. Clear the cache.

Now the URL http://localhost/ez/index.php/plain_site/modul1/create will work. It is the same view as modul1/list but without view parameters.

To store data in our MySQL database, we will build an HTML form that sends the data to our new view (with POST and GET variables). We create a new form in the template *create.tpl* with a text line name. In our example we want to send the data with GET. We also insert a new template variable {$status_message} to show the user a message (see listing 12).

```
{* create.tpl – template for Modulview .../modul1/create
Html form to save the new data *}
<form action={'modul1/create'|ezurl()} method="get">
Name :<br />
<input name="name" type="text" size="50" maxlength="50"><br />
<input type="submit" value="Create new data">
<input type="reset" value="Cancel">
</form>
<hr>
Status: {$status_message}
```

Listing 12. Template jacextension/design/standard/templates/modul1/create.tpl with a form to save the new data.

GET / POST

At this point we change *list.php* to show the GET name variable, which the HTML form transmits to the script. After that we show it in the status field in the template. To show the variable GET / POST we use the eZ Publish framework with the eZHTTPTool class.

First we take a copy of the object of eZHTTPTool with $http =& eZHTTPTool::instance(); With $http->hasVariable('name'); we can discover if the variable $_GET['name'] or $_POST['name'] exists and with $http->variable('name'); we can display it. If only GET or POST variables should be shown, we can use $http->hasGETVariable('name'); or $http->hasPOSTVariable('name');

Information about other functions (for example access to sessions) are described in the API documentation for eZ Publish. For the eZHTTPTool class, see http://pubsvn.ez.no/doxygen/classeZHTTPTool.html.

If we want to write our own log file we use eZLog::write(). This is often useful because the default log files contain a lot of information and are not easy to read (see Listing 13).

```
<?php
// modul1/create.php – Function file of View create
// announce biblioteca php necessary for template functions
include_once( "kernel/common/template.php" );
$Module =& $Params['Module'];

// take copy of global object
$http =& eZHTTPTool::instance ();
$value = '';

// If the variable 'name' is sent by GET or POST, show variable
if( $http->hasVariable('name') )
    $value = $http->variable ('name');

if( $value != '' )
    $statusMessage = 'Name: '. $value;
    else
    $statusMessage = 'Please insert data';

// inicializar Templateobject
$tpl =& templateInit();
$tpl->setVariable( 'status_message', $statusMessage );

// Write variable $statusMessage in the file eZ Debug Output / Log
// here the 4 different types: 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 write own Log file to ezroot/var/log/jacextension_modul1.log
include_once('lib/ezfile/classes/ezlog.php');
eZLog::write ( $statusMessage, 'jacextension_modul1.log', 'var/log');

$Result = array();
// search/replace template and save result for $module_result.content
$Result ['content'] =& $tpl->fetch( 'design:modul1/create.tpl' );

// generate route Modul1/create
$Result ['path'] = array( array( 'url' => 'modul1/list',
                                 'text' => 'Modul1'),
                          array( 'url' => false,
                                 'text' => 'create' ) );
?>
```

Listing 13. jacextension/module/modul1/create.php with examples to show variables GET/POST and generate Debug messages and Log files

Accessing the database

Now we return to the database. We want to save the value of the form in our new table jacextension_data. To do this, we use the eZ Publish class eZPersistentObject. It contains functions for creating, changing, deleting or extracting data.

To use these functions we create a new class JACExtensionData. We save it in the folder *ezroot/extension/jacextension/classes* with the name *jacextensiondata.php* (see Listing 14).

```
<?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 );
    }

    /*!
    Definition of the data object structure /of the structure of the database table
    */
    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'
                      );
    }

    /*!
    Here we extend attribute() of eZPersistentObject
    */
    function &attribute( $attr )
    {
        if ( $attr == 'user_object' )
            return $this->getUserObject();
        else
            return eZPersistentObject::attribute( $attr );
    }

    /*!
    Help function will open in attribute function
    */
    function &getUserObject( $asObject = true )
    {
        $userID = $this->attribute('user_id');
        $user = eZUser::fetch($userID, $asObject);
        return $user;
    }

    /*!
    creates a new object of type JACExtensionData and shows it
    */
    function create( $user_id, $value )
    {
        $row = array( 'id' => null,
                      'user_id' => $user_id,
                      'value' => $value,
                      'created' => time() );
        return new JACExtensionData( $row );
    }
```

Listing 14. jacextension/classes/jacextensiondata.php example for access to the database with eZ PersistentObject

The most important function is JACExtensionData::definition(). This defines the object structure of JACExtensionData, specifying the table and columns where the data will be stored.

Next we create the functions create(), fetchByID(), fetchList(), getListCount(). There are three different types in which the data will be shown:

1. eZPersistentObject::fetchObject()
2. eZPersistentObject::fetchObjectList()
3. direct SQL command

If possible, you should use the functions eZPersistentObject fetch. You should not use special SQL entries for the database, which ensures that the SQL commands will work with all the databases supported by eZ Publish. (See the related API documentation at http://pubsvn.ez.no/doxygen/3.9/html/classeZPersistentObject.html .)

Now we use the new functions in the script *create.php*. To save new data we create a new object of the type JACExtensionData with the function JACExtensionData::create($value) . The function create() creates the transmitted value (value of the form), the current user ID and the current hour.

With the function store() we save the data in our database table jacextension_data. To see this happening we write these to the Debug view. *create.php* is shown below.

```php
<?php
// modul1/create.php – Function file of View create
// ...
$value = '';

// If the variable 'name' is sent by GET or POST, show variable
if( $http->hasVariable('name') )
    $value = $http->variable('name');

if( $value != '' )
{
include_once('kernel/classes/datatypes/ezuser/ezuser.php');

// ask for the ID of current user
$userId = eZUser::currentUserID();

include_once('extension/jacextension/classes/jacextensiondata.php');

// generate new data object
$JacDataObject = JACExtensionData::create( $userId, $value );
eZDebug::writeDebug( '1.'.print_r( $JacDataObject, true ),
                     'JacDataObject before saving: ID not set') ;

// save object in database
$JacDataObject->store();
eZDebug::writeDebug( '2.'.print_r( $JacDataObject, true ),
                     'JacDataObject after saving: ID set') ;

// ask for the ID of the new created object
$id = $JacDataObject->attribute('id');

// ask for the login of the user who has created the data
$userObject = $JacDataObject->attribute('user_object');
$userName = $userObject->attribute('login');

// show again the data
$dataObject = JACExtensionData::fetchByID($id);
eZDebug::writeDebug( '3.'.print_r( $dataObject, true ),
                     'JacDataObject shown with function fetchByID()');

// investigate the amount of data existing
$count = JACExtensionData::getListCount();
$statusMessage = 'Name: >>'. $value .
                 '<< of the user >>'. $userName.
                 '<< In database with ID >>'. $id.
                 '<< saved!New ammount = '. $count ;
}
else
{
    $statusMessage = 'Please enter data';
}

// take data as object and as array and show in 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 )');

// initialize Templateobject
$tpl =& templateInit();
$tpl->setVariable( 'status_message', $statusMessage );

//...

?>
```

Listing 15. jacextension/modules/modul1/create.php. Creating a new entry in the database and different ways of showing it.

To access objects of the type eZPersitentObject, we use $JacDataObject- >attribute('id'). The function parameter “id” maps to the column id in the table, so we don't have to think much to access the different values. This can be applied to all data that is saved in eZ (for example eZContentObject and eZUser objects).

Template fetch function

We now know how to transmit parameters and GET / POST variables to a new view and show them. But if we want to show data (for example our database table) in an eZ Publish template, we cannot handle it just by opening the view.

To do this we use the fetch functions from the "kernel" eZ module, for example {fetch('content', 'node', hash('node_id', 2)}. We want to define two fetch functions, list and count, which are opened by the following template syntax:

```
{fetch( 'modul1', 'list', hash( 'as_object' , true() ) )}

{fetch( 'modul1', 'count', hash() )}
```

The function list shows the data entries as arrays or objects (based on the setting of the parameter ‘as_object’). The count function doesn't have any parameters and uses an SQL command to determine the amount of data in our database table jacextension_data.

The definition of the fetch functions is done in the file *jacextension/modules/modul1/function_definition.php*. This also defines which parameters are transmitted to which PHP function within the specified PHP class.

```
<?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/m
                                                      '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/jacextensio
                                                        'class' => 'eZModul1FunctionCollection',
                                                        'method' => 'fetchJacExtensionDataListCo
                                'parameter_type' => 'standard',
                                'parameters' => array()
                        );

?>
```

Listing 16. Definition of the fetch functions of modul1 - extension/jacextension/modules/modul1/function_defintion.php

The file *jacextension/modules/modul1/ezmodul1functioncollection.php* has a help class that contains all the fetch functions.

```
<?php
include_once('extension/jacextension/classes/jacextensiondata.php');

class eZModul1FunctionCollection
{
    function eZModul1FunctionCollection()
    {
    // ...
    }

    // is opened by('modul1', 'list', hash('as_object', $bool ))
    // fetch
    function fetchJacExtensionDataList( $asObject )
    {
        return array( 'result' => JACExtensionData::fetchList($asObject) );
    }

    // is opened by('modul1', 'count', hash() ) fetch
    function fetchJacExtensionDataListCount( )
    {
        return array( 'result' => JACExtensionData::getListCount() );
    }
}
?>
```

Listing 17. Help class with PHP functions that are used in the Template Fetch definition in function_defintion.php – extension/jacextension/modules/ modul1/ezmodul1functioncollection.php

Hint: In each module, the file *functioncollection.php* lists the possible parameters for the hash() part of a fetch function. Investigate the file *kernel/content/ezcontentfunctioncollection.php* to determine which parameters are available for the fetch function {fetch('content', 'tree', hash(...))} of the kernel content module of eZ Publish. This helps if the eZ documentation is incomplete.

Template operators

Another way to access functions in your extensions is to use template operators. While eZ Publish contains many template operators, we will define a new template operator called "$result_type" with a result_type parameter.

We will use the parameter to control the amount of data that is shown in our database table. The template command {jac('list')} shows an array of data and {jac('count')} shows the amount of data.

We will use the same functions as in the template fetch example: fetch('modul1', 'list' , ...) and fetch('modul1', 'count', ...)

The definition of the existing template operators in the jacextension is done in the file *extension/jacextension/autoloads/ eztemplateautoload.php* (see Listing 18). The template operator's actions are defined in a custom PHP class (in our case, in JACOperator of the file *extension/jacextension/autoloads/ jacoperator.php*) (see Listing 19).

```
<?php

// Which operators will load automatically?
$eZTemplateOperatorArray = array();

// Operator: jacdata
$eZTemplateOperatorArray[] = array( 'script' => 'extension/jacextension/autoloads/jacoperator.ph
                                    'class' => 'JACOperator',
                                    'operator_names' => array( 'jac' ) );
?>
```

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

```
<?php
/*!
 Operator: jac('list') and 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' ))
                    );
    }

    /*!
     Depending of the parameters that have been transmitted, fetch objects JACExtensionData
     {jac('list)} or count data {jac('count')}
    */
    function modify( &$tpl, &$operatorName, &$operatorParameters, &$rootNamespace, &$currentName
    {
        include_once('extension/jacextension/classes/jacextensiondata.php');
        $result_type = $namedParameters['result_type'];
        if( $result_type == 'list')
            $operatorValue = JACExtensionData::fetchList(true);
            else if( $result_type == 'count')
                    $operatorValue = JACExtensionData::getListCount();
    }
}
?>
```

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

To tell eZ Publish that jacextension contains template operators, we have to define it in the eZ Publish configuration file *extension/jacextension/settings/site.ini.append.php* with ExtensionAutoloadPath[]=jacextension (see Listing 20).

```
<?php /* #?ini charset="utf-8"?
...
# search for template operators in jaceextension
[TemplateSettings]
ExtensionAutoloadPath[]=jacextension
*/ ?>
```

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

To check our template fetch functions fetch('modul1', 'list', hash('as_object', true())) and fetch('modul1', 'count', hash()) of the template operator jac('list') or jac('count') we extend the template *list.tpl* of the view list (see Listing 21) .

```
<h2>Template Operator: jac('count') and jac('list')</h2>
Count: {jac( 'count' )} <br />
List: {jac( 'list' )|attribute(show)}
<hr />

<h2>Template Fetch Functions:
fetch('modul1','count', hash() )<br /><br />
fetch('modul1','list', hash( 'as_object', true()) )</h2>
Count: {fetch( 'modul1', 'count', hash() )} <br>
List: {fetch( 'modul1', 'list', hash( 'as_object', true())) | attribute('show')}
```

Listing 21. Testing the custom Template Fetch functions and the template operator - *extension/jacextension/design/standard/templates/modul1/list.tpl*

We open the view by, for example, accessing the URL http://localhost/ez/index.php/plain_site/modul1/list/tableblue/1234

Apart from the example array $data_array, it must show the data of the database table jacextension_data twice, once with the template operators and another time with the template fetch functions. This shows that there are different options for accessing the same functions in your extension in a template.

INI file

Lastly, we want to create our own default .ini file *extension/jacextension/settings/jacextension.ini*. This will store all the values we have set in templates or modules and those that may vary in different eZ Publish installations (for example, for showing special debug messages).

The default .ini can be overwritten by the *jacextension.ini.append.php* files, such as the override file for a siteaccess. Listing 22 is an example of an .ini file and Listing 23 shows how to access it via PHP, as we have extended *list.php*.

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

Listing 22. Configuration file of extension jacextension – extension/jacextension/settings/jacextension.ini

```
<?php
// ...

// read variable JacDebug of INI block [JACExtensionSettings]
// of INI file jacextension.ini
include_once( "lib/ezutils/classes/ezini.php" );

$jacextensionINI =& eZINI::instance( 'jacextension.ini' );

$jacDebug = $jacextensionINI->variable('JACExtensionSettings','JacDebug');

// If Debug is activated do something
if( $jacDebug === 'enabled' )
    echo 'jacextension.ini: [JACExtensionSetting] JacDebug=enabled';

// ...
?>
```

Listing 23. PHP access to INI files jacextension.ini – extension/jacextension/modules/modul1/list.php

Conclusion

With a simple example we have learned several techniques useful for creating eZ Publish extensions.

In addition to creating custom modules with different views and view parameters, template fetch functions and template operators, we also looked at the privilege system of eZ Publish and now know how to write custom messages in the Debug view or in Log files. We also looked at accessing INI files.

With this basic knowledge it should be possible for you to create your own eZ Publish extensions.

You can download the source code for the tutorial at: http://ez.no/community/contribs/examples/jacextension

Resources

- http://www.ezpublish.de/ – German eZ community
- http://www.ez.no/community - International eZ community
- http://pubsvn.ez.no/doxygen/index.html – eZ API documentation
- http://ez.no/doc/ez_publish/technical_manual/3_9/reference – eZ reference documentation
- http://ez.no/community/contribs/documentation/jac_dokumentation_in_german_ez_publish_basics_extension_development – PDF eZ publish basics in programming modules (in German)
- http://ez.no/de/developer/articles/an_introduction_to_developing_ez_publish_extensions - This tutorial in German
- http://ez.no/developer/contribs/documentation/jac_tutorial_ger_de_ez_publish_extension_entwicklung - This tutorial in Spanish
- http://ez.no/community/contribs/examples/jacextension - Source code for the tutorial