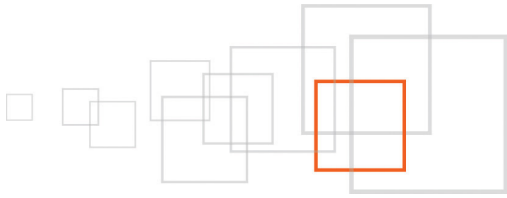
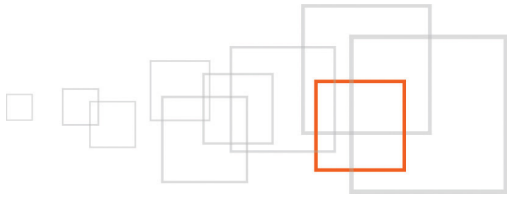


Fetching User Objects with PHP: Part 2

By David Linnard
www.onequarterenglish.co.uk



Index

1 Goal description

At the end of this tutorial you should be comfortable with exporting multiple users from PHP scripts into a external file using a cronjob. You should also be comfortable with filtering fetch statements in PHP.

2 Introduction

There are specific cases where you need to pull the eZ Publish information directly in your PHP scripts. There are options available for both users and nodes. This tutorial provides a detailed guide about extracting user Information through a script and going through some practical examples.

Part One of the tutorial provided ways to extract individual users and demonstrated how to display all of the user information. It also demonstrated some of the built in ways you can extract multiple users.

This part of the tutorial concentrates on the eZContentTreeNode::subTreeByNodeID() method. We will look at its basic use and some practical examples before creating a cronjob to automate the process.

3 Pre-requisites and target population

You should have a 4.x build of eZ Publish, be comfortable with the structure of eZ Publish source files and be able to run scripts through the command line. Knowledge of the cronjob functionality is also required. Being comfortable with eZ Publish scripts are advantageous. You should also have read the first part of the tutorial.

4 Filtering Users using subTreeByNodeID

We've looked at a few ways of pulling out multiple users but by far the most useful is achieved by using Fetch statements. We're going to be using the function eZContentObjectTreeNode::subTreeByNodeID() as this is extremely flexible and can be adapted to suit whatever content classes you need to export or access in your PHP.

Below is a fairly simple example using the function. It pulls out the first 5 users found on the site within Users/Members (we'll break the script down once we've looked at it).

The example assumes you already seen the first part of the tutorial, this will give you a basic script you can house the rest of the code examples in.

```
$offset = 0; //Offset defaults to zero if omitted

$limit=5; //lets limit the fetch to 5 to prevent too many results from being returned

$depth = 1; //Depth defaults to one if it is excluded

$includeClasses = array( 'user' ); //please note this refers to content classes created
in the CMS, rather than PHP files


$params = array( 'Depth' => $depth,

                          'Offset' => $offset,
```

```
                            'Limit' => $limit,
                            'ClassFilterType' => 'include',
                            'ClassFilterArray' => $includeClasses);


// getting the ID of where the users sit in the CMS (limiting the area eZ has to search
for the objects):
$parent_node = eZContentObjectTreeNode::fetchByURLPath( 'users/members' );
$parent_node_id = $parent_node->attribute( 'node_id' );
$results = eZContentObjectTreeNode::subTreeByNodeID( $params, $parent_node_id );
```

4.1 The fetch statement

There are a few areas to pay attention to in this example so we'll break it down and look at each area. We will start at the end since this is the function call itself. We will then look at what we are passing in to the function:

```
…
$results = eZContentObjectTreeNode::subTreeByNodeID( $params, $parent_node_id );
```

The code is equivalent to the following fetch statement (parameters are specified for completeness):

```
{def $fetch_nodes=fetch( 'content', 'list', hash( 'parent_node_id', 2, 'offset', 0,
'limit', 10) )}
```

Unlike the fetch statement where a single associative array contains both the parent node ID and the other possible parameters, the PHP version separates the two. Our first parameter can contain filters, offsets and limits while our second parameter is solely the parent node ID.

4.2 The parameter array

```
…
$offset = 0; //Offset defaults to zero if omitted
$limit=5; //lets limit the fetch to 5 to prevent too many results from being returned
$depth = 1; //Depth defaults to one if it is excluded
$includeClasses = array( 'user' ); //please note this refers to content classes created
in the CMS, rather than PHP files


$params = array( 'Depth' => $depth,
                            'Offset' => $offset,
                            'Limit' => $limit,
                            'ClassFilterType' => 'include',
                            'ClassFilterArray' => $includeClasses);
…
```

This section of the code is split into two parts. The first part is used to set the variables and the second just

stores these in an array. Below is a complete list of the parameters you can pass into the system. They work in the same way as the parameters do in the fetch statement. Our following examples will include the other parameters so you should have a fully working example of what you need to export by the end of the tutorial.

Parameter	About	Example
Depth	By default set to 1, this value tells eZ how many levels it should search down.	array('Depth' => 3)
Offset	By default set to 0, this value is usually used in conjunction with Limit to control the offset of the results.	array('Offset' => 20)
Limit	By default all of the results are returned. This can be set in conjuction with Offset to make your queries more resource friendly.	array('Limit' => 20)
SortBy	This can be used to sort in order of name, published date or any attribute values. You must use true/false to specify ASC/DESC.	array('Limit' => array("name", true))
AttributeFilter	This is the key parameter which we will look into in more detail shortly. For user export you may use this in a number of ways, whether to check users who wish to be contacted or users created within a specified timespan. Our example here is simple. We will look at more complex examples but I would also recommend looking at the content list documentation on the eZ.no site for more examples.	array('AttributeFilter' => array(array('name', 'like', 'David*')))
ExtendedAttributeFilter	This is a topic in itself so it is not covered here. Check out the examples in the fetch documentation if you need to use this with your custom queries.	

ClassFilterType / ClassFilterArray	These are used in tandem so we will show their use together. ClassFilterType can be set to include or exclude. The classes filtered relate to the content types created in the CMS (such as User and Article) and do not relate to physical PHP file types (like eZUser). Please note that you need to use the class identifier rather than the class name.	Array('ClassFilterType' => 'include', 'ClassFilterArray' => array('user'))
GroupBy	Deprecated and not really that useful. Check out the codebase of eZContentObjectTreeNode if you want to use this.	

4.3 The Parent Node ID

```
$parent_node = eZContentObjectTreeNode::fetchByURLPath( 'users/members' ); //note the
lowercase. URLs use underscores rather than hyphens

$parent_node_id = $parent_node->attribute( 'node_id' );
```

There are a couple of points to mention here. Although we could easily hard code a value for the parent node id in our code I prefer not too for code readability and because there is less chance of making a mistake. Note the use of lowercase and also be wary that you change any hyphens to underscores (for example My-Content would become my_content).

Finally, be aware that a lot of our examples in part one of this tutorial have used the Content Object ID whereas here we need to use the **Node ID**.

5 Practical Examples

Let's look at some more practical uses for our fetches. Let's assume we are working for a company who's marketing department are very keen on analysing and utilising our site's user data. They've made some requests about data they would like from the system:

1. A complete list of site users and the total number of users
2. The total number of new users last month and who those users were
3. The total number of users who modified their account last month (apart from the new users)
4. The number of users who wish to be contacted by email and who these users are

First of all, that is a lot of requests! Luckily though, the code required is very similar. We are going to look at these in turn before creating a cronjob which will pull together a lot of what we have looked at so far.

The key assumption in the following code examples is that our month runs from the 1st to the end of the month.

We are also assuming that the new month will have started when the code is run (even if it only started a couple of minutes ago).

5.1 A complete list of site users and the total number of users

It would be easy to do this using the eZRole:: fetchUserByRole() function to extract users by role however there can potentially be a couple of issues with doing this:

- If you have a very large number of users and you are loading content into a template file your page could timeout
- If you need to use more than one role you need to carry out multiple queries

Due to these limitations, and also to look at how we would do this using eZContentObjectTreeNode::subTreeByNodeID, we will use this instead.

The code for this is actually pretty similar to our basic example. Since we know only site users and not admin users will be included within this folder, we can remove the limit we imposed and return all of the:

```
$includeClasses = array( 'user' ); //please note this refers to content classes created
in the CMS, rather than PHP files


$params = array( 'ClassFilterType' => 'include',
                     'ClassFilterArray' => $includeClasses);


// getting the ID of where the users sit in the cms (limiting the area eZ has to search
for the objects):
$parent_node = eZContentObjectTreeNode::fetchByURLPath( 'users/members' );
$parent_node_id = $parent_node->attribute( 'node_id' );
$all_users = eZContentObjectTreeNode::subTreeByNodeID( $params, $parent_node_id );
```

Getting the count is straightforward. There is a comparable function to subTreeByNodeID which exports the count rather than the data itself. It can be used in exactly the same way and so we can add the following line to the bottom of our previous code to get the total number:

```
...
$all_user_count = eZContentObjectTreeNode::subTreeCountByNodeID( $params,
$parent_node_id );
```

5.2 The total number of new users last month and who those users were

We now need to extend the previous example by ensuring only people from the last month are included. We are going to work out the start and end of the month and ensure that we only extract users created within these two dates. We can do this by making use of an attribute_filter, just as we would do in a Fetch statement within our templates .

Our attribute filter is as follows. Some straightforward PHP is used to work out the start and end dates. If you wanted to extract information from a different timespan then you could easily modify this code. For example, you may want to export the information on a weekly basis.

```
//working out the month start and end dates for the previous month (assuming the month runs from the 1st to the end of the month), code compatible with all versions of PHP 5:

$first_of_month = strtotime( date( "Y-m-1" ) ); //this is our endtime, midnight on the first of the current month

$first_of_last_month = strtotime( '-1 month' , $first_of_month ); //our start time is one month before then.


$attributeFilter  = array( array( 'published', 'between', array( $first_of_last_month, $first_of_month) ) ); //our attribute filter is ready to add to our parameters
```

All we need to do now is add the attribute filter we have created to our previous example which should give us the following code:

```
$includeClasses = array( 'user' );

$sortBy = array( "name", true ); //let's sort by name


//working out the month start and end dates for the previous month, compatible with all versions of PHP 5:

$first_of_month = strtotime( date( "Y-m-1" ) ); //this is our endtime, midnight on the first of the current month

$first_of_last_month = strtotime( '-1 month' , $first_of_month ); //our start time is one month before then.


$attributeFilter  = array( array( 'published', 'between', array( $first_of_last_month, $first_of_month ) ) );


$params = array( 'SortBy' => $sortBy,

                         'ClassFilterType' => 'include',

                         'ClassFilterArray' => $includeClasses,

                         'AttributeFilter' => $attributeFilter);


//getting the ID of where the users sit in the cms (limiting the area eZ has to search for the objects):

$parent_node = eZContentObjectTreeNode::fetchByURLPath( 'users/members' ); //note the lowercase. Use underscores rather than hyphens if spaces are included in the path

$parent_node_id = $parent_node->attribute( 'node_id' );


$new_users = eZContentObjectTreeNode::subTreeByNodeID( $params, $parent_node_id );

$monthly_new_user_count = eZContentObjectTreeNode::subTreeCountByNodeID( $params, $parent_node_id );
```

5.3 Users who modified their last month account (apart from the new users)

This example is slightly more complicated. We want to extract users who were modified in the last calendar month but this can not include users who were created. Recall that fetch statements can have multiple attribute filters. To carry out our query we need to ensure two things:

- The user was not created between the first and last dates of last month
- The user was modified between the first and last dates of last month

In our last example we used "between" to extract the users who were published within the previous month so to exclude those users we can modify the code to be "not_between". We can then use "between" on the modified date to ensure we pick up the modified users. The following code should do the trick:

```
//working out the month start and end dates for the previous month, compatible with all versions of PHP 5:
$first_of_month = strtotime( date( "Y-m-1" ) ); //this is our endtime, midnight on the first of the current month
$first_of_last_month = strtotime( '-1 month' , $first_of_month ); //our start time is one month before then.


$attributeFilter  = array( array( 'modified', 'between', array( $first_of_last_month, $first_of_month ) ),
                                                array( 'published', 'not_between', array( $first_of_last_month, $first_of_month ) ) );
```

As you can see the only addition to the attribute_filter is an additional array element. For a fully functional example, we can modify our previous code to the following:

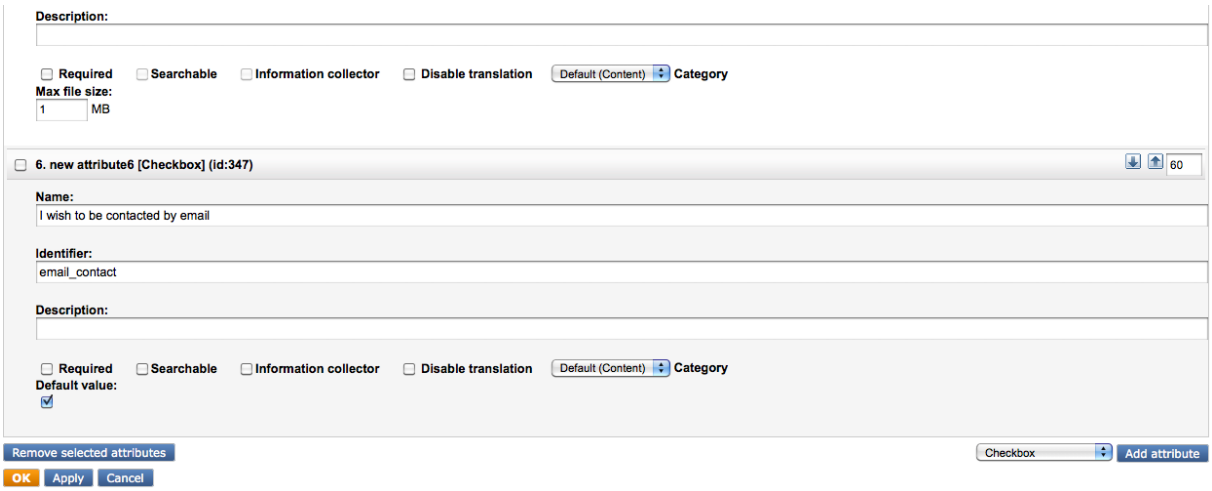
```
$includeClasses = array( 'user' );
$sortBy = array( "name", true ); //let's sort by name


//working out the month start and end dates for the previous month, compatible with all versions of PHP 5:
$first_of_month = strtotime( date( "Y-m-1" ) ); //this is our endtime, midnight on the first of the current month
$first_of_last_month = strtotime( '-1 month' , $first_of_month ); //our start time is one month before then.


$attributeFilter  = array( array( 'modified', 'between', array( $first_of_last_month, $first_of_month ) ),
                                                array( 'published', 'not_between', array( $first_of_last_month, $first_of_month ) ) );


$params = array( 'SortBy' => $sortBy,
                            'ClassFilterType' => 'include',
                            'ClassFilterArray' => $includeClasses,
```

```
                        'AttributeFilter' => $attributeFilter);



//getting the ID of where the users sit in the cms (limiting the area eZ has to search
for the objects):

$parent_node = eZContentObjectTreeNode::fetchByURLPath( 'users/members' ); //note the
lowercase. Use underscores rather than hyphens if spaces are included in the path

$parent_node_id = $parent_node->attribute( 'node_id' );


$modified_users = eZContentObjectTreeNode::subTreeByNodeID( $params, $parent_node_id );

$monthly_modified_user_count = eZContentObjectTreeNode::subTreeCountByNodeID( $params,
$parent_node_id );
```

5.4 The number of users who wish to be contacted by email and who these users are

Many commercial sites need content such as this and you may well require multiple fields (e.g. contact by third parties, contact by phone, contact by post etc). When you start adding custom fields to the user class I would suggest creating a new content class as the content will not be required for some user types (admin users for instance). In this case though for simplicity we will add a new field to the standard user type.

Go to the Setup in the CMS and then click on the "Classes" link. Within the Class Groups you should see "Users" listed so click on this. If we were creating a new user type here we can choose to copy the existing user type and add new fields to the new class. Since we are not just click on the pencil symbol to edit the class. Once you do so go to the bottom of the class definition and you should be given an option to add an attribute (below is what you see in eZ Publish 4.3 where it is located in the bottom right but in previous versions it has been in the bottom left):



Create a field similar to the one in the screenshot. The details for it are as follows:

- **Field Type:** checkbox
- **Field Name:** I wish to be contacted by email

- **Identifier:** contact_by_email
- **Checkboxes:** All Unchecked

Now create a user and make sure your new checkbox is checked, otherwise we will have no results.

We now need to write an attribute filter that checks for users who wish to be contacted. This will work in the same way as it does in template files. The following code we do what we need:

```
$attributeFilter  = array( array( 'user/contact_by_email', '=', 1 ) );
```

We don't need to check any dates in this case so our final code is as follows:

```
$includeClasses = array( 'user' );
$sortBy = array( "name", true ); //let's sort by name


$attributeFilter  = array( array( 'user/contact_by_email', '=', 1 ) );


$params = array( 'SortBy' => $sortBy,
                         'ClassFilterType' => 'include',
                         'ClassFilterArray' => $includeClasses,
                         'AttributeFilter' => $attributeFilter);



//getting the ID of where the users sit in the cms (limiting the area eZ has to search
for the objects):
$parent_node = eZContentObjectTreeNode::fetchByURLPath( 'users/members' ); //note the
lowercase. Use underscores rather than hyphens if spaces are included in the path
$parent_node_id = $parent_node->attribute( 'node_id' );


$users_to_email = eZContentObjectTreeNode::subTreeByNodeID( $params,$parent_node_id );
$users_to_email_count = eZContentObjectTreeNode::subTreeCountByNodeID( $params,
$parent_node_id );
```

6 Putting it all together

Let's put all of what we have covered together and create a full cronjob script for exporting user information that can be run on a monthly basis. We will export all new users created by the system each month and store it in a tab delimited text file (we will not use a CSV as there is a good chance that if you are storing user address fields that they will contain commas). We will also send an email confirmation that the file has been created which will include stats on the number of new site users, the total number of active site users and the number of users who wish to be emailed.

Before we begin we need to make sure a directory exists to store our exports. Create a directory in your var folder called "user_exports" (also make sure eZ Publish can write to the directory).

6.1 Formatting the user data

To make our code more legible we will move the code we created in part one of this tutorial to display user details into a separate function. We will also modify the information so it is shown in a format suitable for a tab delimited file. We will be using the eZContentObjectTreeNode::subTreeByNodeID() function to extract users. Due to this we will be dealing with eZContentObjectTreeNodes and so the code in the function will reflect this:

```
function show_user( $userNode )
{
	// first add the id for reference:

	$return_string = $userNode->attribute( 'node_id' );

	// now extract the fields:

	$userObj  = $userNode->attribute( 'object' ); //we will handle pulling the
information out of these later.
	$dataMap = $userObj->attribute( 'data_map' );

	//print each in turn:
	foreach( $dataMap as $key => $value )
	{
		$type = $value->dataType();
		switch( $type->DataTypeString )
		{
			case 'ezuser':
				$user_account = $dataMap['user_account']->attribute( 'content'
);
				$return_string .= "\t{$user_account-
>attribute( 'login' )}\t{$user_account->attribute( 'email' )}";
				break;
			case 'ezstring': //for basic text & ints
			case 'eztext':
			case 'ezint':
			case 'ezfloat':
				$return_string .= $value->toString();
				break;
			case 'ezimage':
				$content = $value->attribute( 'content' );
				$displayText = $content->displayText();
				$imageAlias = $content->imageAlias( 'original' );
```

```
                        $imagePath = $imageAlias['url'];
                        $return_string .= "\t$displayText ($imagePath)";
                        break;
                }
        }
        $return_string .= "\n";
        return $return_string;
}
```

We also need to create a header row for our data export. To make things easy for us, let's modify the code we have just created and display the key for each field instead of the field value itself:

```
function show_header($userNode)
{
        $return_string = "Node ID";

        $userObj  = $userNode->attribute( 'object' ); //we will handle pulling the
information out of these later.
        $dataMap = $userObj->attribute( 'data_map' );

        foreach( array_keys( $dataMap ) as $key )
        {
                $return_string .= "\t$key";
        }
        $return_string .= "\n";
        return $return_string;

}
```

6.2 Fetching the user information

Now that we have the functions created to show a header and to show a user, let's put them into use and store details of all new users into a variable:

```
$includeClasses = array( 'user' );
$sortBy = array( "name", true ); //let's sort by name

//working out the month start and end dates for the previous month, compatible with all
versions of PHP 5:
$first_of_month = strtotime( date( "Y-m-1" ) ); //this is our endtime, midnight on the
first of the current month
$first_of_last_month = strtotime( '-1 month' , $first_of_month ); //our start time is one
```

```
month before then.


$attributeFilter  = array( array( 'published', 'between', array( $first_of_last_month,
$first_of_month ) ) );


$params = array( 'SortBy' => $sortBy,
                          'ClassFilterType' => 'include',
                          'ClassFilterArray' => $includeClasses,
                          'AttributeFilter' => $attributeFilter);



//getting the ID of where the users sit in the cms (limiting the area eZ has to search
for the objects):
$parent_node = eZContentObjectTreeNode::fetchByURLPath( 'users/members' ); //note the
lowercase. Use underscores rather than hyphens if spaces are included in the path
$parent_node_id = $parent_node->attribute( 'node_id' );


$new_users = eZContentObjectTreeNode::subTreeByNodeID( $params, $parent_node_id );
$monthly_new_user_count = eZContentObjectTreeNode::subTreeCountByNodeID( $params,
$parent_node_id );


/*print header info:*/
$file_content = show_header( $new_users[0] );


foreach( $new_users as $user )
{
  $file_content .= show_user( $user );
}
```

We can pull out the total number of users who have modified their details and the number of users who wish to be emailed using the code we have already looked at. I will omit repeating the code here so we can move on to the other functionality required (the full source code is available at the end of the tutorial).

6.3 Storing the content to file

The first bit of additional functionality we need is to store our variable to a file. We'll do this with some standard PHP code:

```
//creating the file (filename based on start and end dates of the data):
$file_name = date( 'd-m-Y', $first_of_last_month ) . '_' . date( 'd-m-Y', strtotime( '-1
day' , $first_of_month )) . ".txt";
$var_path = $_SERVER['PWD'] . '/var/user_export/';
$file_path = $var_path.$file_name;
```

```
// creating the file and storing our content:
$file_pointer = fopen( $file_path, 'x' );
fwrite( $file_pointer, $file_content);
```

6.4 Sending a notification email

We now need to notify someone that the file has been stored. You will be aware that the site admin's email is available to us through the site.ini file. Since we can extract it from here, let's use this as our email address:

```
// pulling out the site admin's email address for sending the email:
$site_ini = eZIni::instance( "site.ini" );
$admin_email =$site_ini->BlockValues['MailSettings']['AdminEmail'];
```

Now we have the admin's email, let's create our email message and sent it to them. Let's include the stats so they have details of the latest export:

```
//sending the email containing user stats:
$email_subject = "Users report for" . Date( 'd-m-Y', $first_of_last_month ) . ' to ' .
date( 'd-m-Y', strtotime( '-1 day' , $first_of_month ) );
$email_content = 'The New Users Report from the start of  ' . date( 'd-m-Y',
$first_of_last_month ) . ' to the end of ' . date( 'd-m-Y', strtotime( '-1 day' ,
$first_of_month ) ) . ' has been created at: ' . $file_path;
$email_content .= "\n\nHere is a summary of the user stats:\n\n";
$email_content .= 'New Users: ' . $monthly_new_user_count . "\n";
$email_content .= 'Modified Users: ' . $monthly_modified_user_count . "\n";
$email_content .= 'Users who wish to be emailed: ' . $users_to_email_count . "\n";
//sending our email with a link to our file to the site admin:
mail( $admin_email, $email_subject, $email_content ); //we don't need any real styling so
let's just use the standard php email function
```

7 Conclusion

And that is it! If you open up the file in a spreadsheet application you should be able to open it so that each tab is shown in a separate column. The script we have created is flexible and you could pull out any kind of content based on the principles we have looked at over these tutorials, just by modifying the Fetch statement we are using. The complete code is downloadable so make sure to play around with it so that it works for you.

8 Resources

- First part of this tutorial : http://share.ez.no/learn/ez-publish/fetching-user-objects-with-php-part-1
- http://serwatka.net/blog/fetching_ez_publish_content_objects_with_php - Fetching eZ Publish content objects with PHP
- http://pubsvn.ez.no/doxygen/trunk/html/classeZContentObjectTreeNode.html - eZContentObjectTreeNode Class reference

- http://pubsvn.ez.no/doxygen/4.0/html/classeZUser.html - eZUser Class reference
- http://pubsvn.ez.no/doxygen/4.0/html/classeZImage.html - eZImage Class reference
- http://ez.no/doc/ez_publish/technical_manual/4_0/reference/modules/content/fetch_functions/list - Template Fetch Functions

9 About the author : David Linnard

David is a London based web developer with a wide variety of skills who has spent the past two years developing for some big eZ Publish commercial sites. He won the eZ Systems 2010 tutorial of the year award for his previous tutorial on the site and was also nominated for Blogger of the year at the same awards. He is also experienced at handling a variety of other content management systems and the Zend Framework.

10 License choice