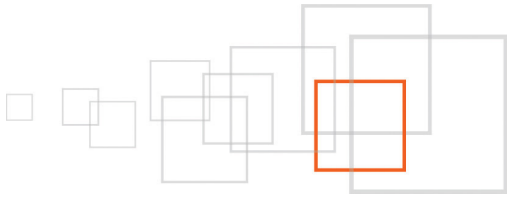
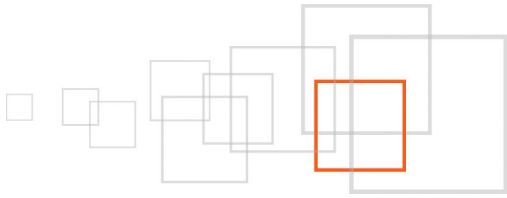


Fetching User Objects with PHP: Part 1

By David Linnard
www.onequarterenglish.co.uk



Index

1 Goal description

At the end of this tutorial you should be comfortable with exporting user information from PHP scripts

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. There is already an excellent article about doing this here. This guide extends this by providing a detailed guide to extracting User Information through a script and going through some practical examples.

This tutorial is split into two parts. In this part, we will look at how you can retrieve individual users from the eZUser class and how to extract information from the eZUser class. I will then show you how to use some of the functions within eZUser and eZRole to extract multiple users.

The second part of this tutorial will detail how to extract user information through eZContentTreeNode::subTreeByNodeID(). As well as demonstrating this function, it will provide practical examples of its use and provide you with a cronjob you can use for your own user exports.

3 Pre-requisites and target population

You should have a 4.x build of eZ Publish and be comfortable with the structure of eZ Publish source fi les and be able to run scripts through the command line. Knowledge of the cronjob functionality and eZ Publish scripts are advantageous.

4 Step 1: Setting up a script

Before we start we need a basic PHP script which will handle the import. Although it is just a PHP script, let's create it within a directory for cronjobs so that the script can be automated when it is fi nished. Please note debug is turned on in the script below so make sure to turn it off when you are happy with it.

```
<?php
set_time_limit ( 0 ); //ensure the script does not time out
require 'autoload.php'; //make sure relevant eZ Classes can be loaded


$cli = eZCLI::instance();//provides interface with CLI


//Setting up the script object itself:
$script = eZScript::instance( array('description' => "eZ Publish user export.\n\n" .
                                                      "Methods of exporting user
information from eZ Publish \n" .
                                                      "\n",
```

```
                                    'use-session' => false,
                                    'use-modules' => true,
                                    'use-extensions' => true,
    'debug-output' => true,
    'debug-message' =>true

                                    ) );
$script->startup();
$script->initialize();


/*ensuring the current user has the rights to the user information:*/
$user = eZUser::fetchByName( 'admin' );
eZUser::setCurrentlyLoggedInUser( $user, $user->attribute( 'contentobject_id' ) );


/**********************
Export User Info Here
**********************/

$script->shutdown(); //stop the script
?>
```

For all code we create, we'll just need to replace the comment block "Export User Info Here" with the code.

You may notice from this example we have already pulled out a user. We need to make sure the current user has the right to view user information in the CMS. By default, the anonymous account is used in the script which will not have access to this information. We therefore change the current user to be admin so that the person does have rights. We will now look at all the ways we can pull individual users out of the CMS.

5 Step 2: Fetching individual users

A specific user can be pulled out of the system by Object ID, email address or their username (as we just did). We can also pull out the user based on the Node ID, but this means extracting the user information has to be approached slightly differently (we will cover this later in the tutorial). Below we use each method to pull out the same user. Since we have just pulled out the admin user, we will do the same here. If you are statically using a user in your script, I would advise using the email address or username where possible for clarity.

```
$users['name'] = eZUser::fetchByName( 'admin' );
$users['email'] = eZUser::fetchByEmail( 'admin@admin.com' );
$users['object_id'] = eZUser::fetch(14);
```

```
print_r( $users );
```

You should see from the results that they all return eZUser Objects.

```
unknown-00-25-00-f9-a9-5c:ezPublications d_linnard$ php cronjobs/exportUserConte
nt.php
Array
(
    [name] => eZUser Object
        (
            [Login] => admin
            [Email] => admin@admin.com
            [PasswordHash] => db2a4cb0603659cc0efd0d7c13084bee
            [PasswordHashType] => 2
            [Groups] =>
            [OriginalPassword] =>
            [OriginalPasswordConfirm] =>
            [PersistentDataDirty] =>
            [ContentObjectID] => 14
        )

    [email] => eZUser Object
        (
            [Login] => admin
            [Email] => admin@admin.com
            [PasswordHash] => db2a4cb0603659cc0efd0d7c13084bee
            [PasswordHashType] => 2
            [Groups] =>
```

5.1 Fetching the current user

If we were not running a standalone script and were instead adding some functionality to the CMS, we may need to pull out the currently logged in user. This can be done easily using the code below. Since we are using an automated script, we will not use this method further here (since this will return an anonymous user):

```
$user = $users['name'];
$cli->output( 'Username: ' . $user->attribute('login') );
$cli->output( 'Email: ' . $user->attribute('email') );
```

6 Step 3: Extracting user information

Now that we have a few user objects to play with, let's look at how we can use pull out the user information

from them.

6.1 Information fetched from the eZUser object

You will be able to see from the script output above we can pull out the login name, content object ID and email directly from the eZ User object. Instead of doing this, we will pull them out using the attribute() method. We are doing this because it follows the way objects are accessed within template files and there is less chance of this functionality changing in future releases. It is therefore good practice to do it in this way. We can find out which attributes can be accessed by using the definition() function of a class. For eZUser, we get the following attributes (please note these two functions are available for all eZContentObjects you may want to export):

- contentobject_id
- login
- email
- password_hash
- password_hash_type

And here's the code we can use to extract the information. Please note that since we are currently only pulling out the same user multiple times we will just use one of the array elements for the next code examples. Later we will add code to extract information from multiple users.

```
$user = $users['name']; //let's just use one of the user's we have extracted

// print_r( $user->definition() ); //getting the definition of what attributes we can
extract. Uncomment to see for the definition for yourself

// printing out the attributes we can get directly from the eZUser object:

$cli->output( 'Username: ' . $user->attribute( 'login' ) );

$cli->output( 'Email: ' . $user->attribute( 'email' ) );

$cli->output( 'Content Object ID: ' . $user->attribute( 'contentobject_id' ) );
```

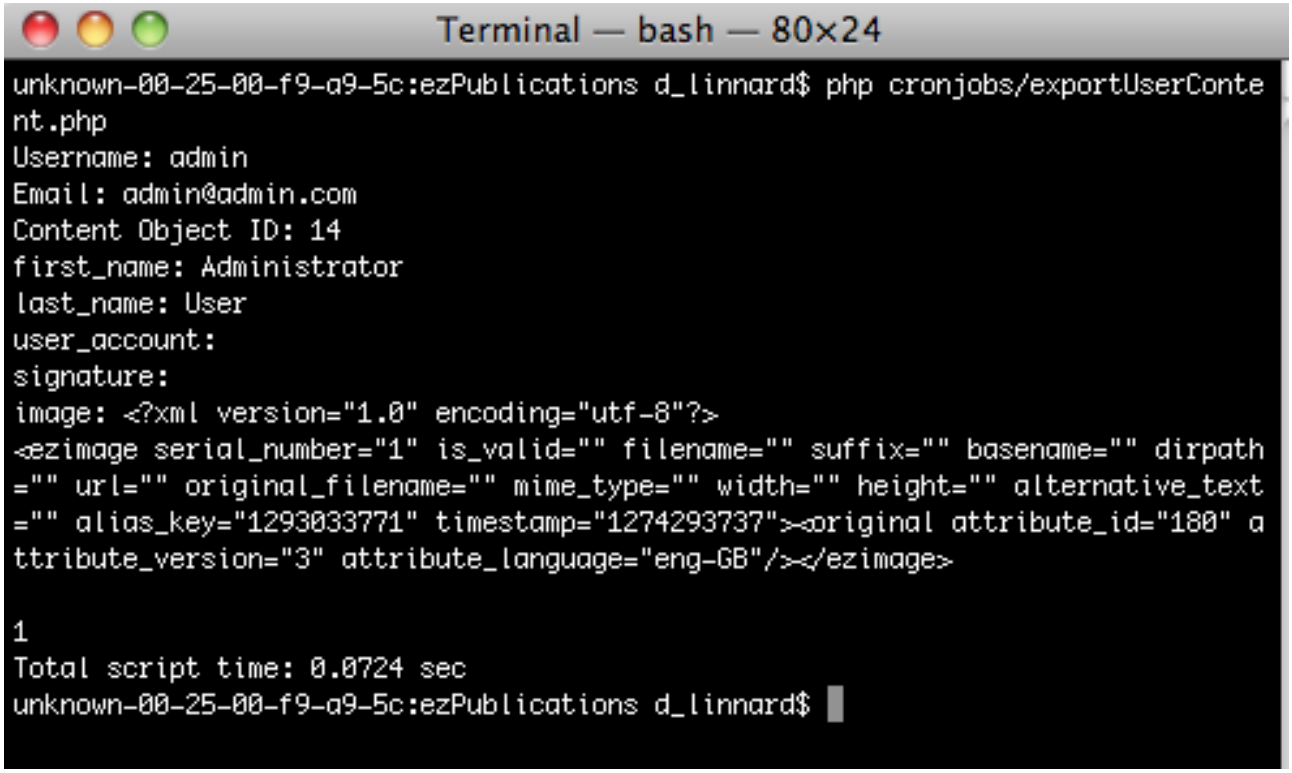
6.2 Fetching other user fields

Pulling out the other user fields is trickier and is hard to do with the eZUser class. Luckily, the eZContentObject class allows you to pull out the DataMap of an object in an associative array. We can convert an eZUser class into an eZContentObject class really easily using the contentObject() function of the eZUser class. Once we have done this we can then extract the data map:

```
// printing out other user fields:

$contentObject = $user->attribute( 'contentobject' ); //converting it to a
eZContentObject using the attribute method


// extracting the dataMap:

$dataMap = $contentObject->dataMap();
```

```
foreach( $dataMap as $key => $value ) //let's start off simply and just print off each
value:
{
       $cli->output( "$key: $value->attribute( 'data_text' )" );
}
```

The script should produce the following output:



As you can probably see from the example, even when you run the script with the default User class, there are a couple of things to look out for :

6.2.1 Correctly displaying images and numbers

Although the majority of the user details are displayed properly, some are not. Before we start let's have a quick recap of how template logic can be different according to different types of fi eld:

```
Name: {$node.data_map.name.data_text}
Price: {$node.data_map.price.data_float}
In Stock: {$node.data_map.no_in_stock.data_int}
```

This is also the case when you extract the information in PHP. The example code we have been using is simplified in that we have assumed every item can be returned as a string (or DataText as it is known in eZ.) Depending on the type of your variable you will want to display, you need to display it in the same way you

would as if you were in the template.

For basic types (for instance strings and numbers, we can make use of the toString() method of each attribute (each item we look at is an eZContentObjectAttribute which provides this automatically). For other fi elds though, we may want to carry out more fi ne tuning. In our user example, there are two such fi elds. We will cover the user_account fi eld shortly but in the meantime let's take a look at how we can display our image in a helpful format.

For the image, as with most complex types, we have more than one possible value we can display. In this case possible values we may want to show are an image path, the image itself or the alternate text. Since we're working through the command line we can't show the image itself so let's show the other two instead. Although simple information can often be extracted straight from the eZContentObjectAttribute, in cases where it is not you can use the content() method which will return the attribute as it sits natively in eZ Publish. In the case of the image, the content will return an ezimage. Looking at the eZImage class reference, these are quite simple to pull out. Let's take a look at an extending our example from above but displaying content based on the content type. By doing this we can then easily extend when we need to extract additional fi eld types (as we will need to shortly) :

```
// printing out other user fields:
$contentObject = $user->attribute( 'contentobject' );
$dataMap = $contentObject->attribute( 'data_map' );

foreach( $dataMap as $key => $value ) //looping through each field
{
	$type = $value->dataType(); //looking at what type the current field is
	switch( $type->DataTypeString ) //base the switch on the type name
	{
		case 'ezimage':
			$content = $value->attribute( 'content' );
			$displayText = $content->displayText();
			$imageAlias = $content->imageAlias('original');
			$imagePath = $imageAlias['url'];
			$cli->output( "$key: $displayText ($imagePath)" );
			break;
		case 'ezstring': //for basic text & ints
		case 'eztext':
		case 'ezint':
		case 'ezfloat':
			$cli->output( "$key: ".$value->toString() );
			break;
		default: //by default let's show what the type is (along with the toString
representation):
			$cli->output( $key.' ' . $type->DataTypeString . ' - ' . $value-
>toString() );
```

```
                break;
        }
}
```

6.2.2 ***Displaying user_account***

The other field of note is called user_account. In the last example if you run the code it will tell you this is an eZUser object. In the cases we have been looking at so far we have always started with a eZUser object and this merely returns the same object we started with. That being the case, why is this useful? Well mostly because you won't always be dealing with eZUser objects in the fi rst place. The second part of this tutorial will look at fetches using a method in eZContentObjectTreeNode. This method will return eZContentObjectTreeNodes, which means we can not access the eZUser attribute method directly to give us the user's email address and username. In this situation we can instead pull this information from the user account field (if you are working with eZContentObjects you will need to do something similar).

The following code adds to the above to ensure user accounts are dealt with :

```
// printing out other user fields:
$contentObject = $user->attribute( 'contentobject' );
$dataMap = $contentObject->attribute( 'data_map' );

foreach( $dataMap as $key => $value ) //looping through each field
{
        $type = $value->dataType(); //looking at what type the current field is

        switch( $type->DataTypeString ) //base the switch on the type name
        {
                case 'ezuser':
                        $user_account = $value->attribute( 'content' );
                        $cli->output( 'Username: ' . $user_account->attribute('login') );
                        $cli->output( 'Email: ' . $user_account->attribute('email') );
                break;
                case 'ezimage':
                        $content = $value->attribute( 'content' );
                        $displayText = $content->displayText();
                        $imageAlias = $content->imageAlias('original');
                        $imagePath = $imageAlias['url'];
                        $cli->output("$key: $displayText ($imagePath)");
                        break;
                case 'ezstring': //for basic text & ints
                case 'eztext':
```

```
            case 'ezint':
            case 'ezfloat':
                  $cli->output( "$key: " . $value->toString() );
                  break;
            default: //by default let's show what the type is (along with the toString
representation):
                  $cli->output( $key . ' ' . $type->DataTypeString . ' - ' . $value-
>toString() );
            break;
      }
}
```

6.3 An aside: Extracting user information from nodes and content objects

We just looked at extracting user account information from the data map rather than from the eZUser object directly. Let's now take a quick look at a standalone example of dealing with extracting this information if we only have a node ID to illustrate this technique :

```
$userNode = eZContentObjectTreeNode::fetch( 15 ); //simple fetch by node id
$userObj = $userNode->attribute( 'object' );
$dataMap = $userObj->attribute( 'data_map' );

$user_account = $dataMap['user_account']->attribute( 'content' );
$cli->output( 'Username: ' . $user_account->attribute( 'login' ) );
$cli->output( 'Email: ' . $user_account->attribute( 'email' ) );
```

To handle this we convert the eZContentObjectNode into an eZContentObject and we can then pull out the content from the datamap as we have just done. I would recommend using the switch statement in the preceding example to pull this information out.

If you are dealing with a eZContentObject, you simply need to use the code from the 3 rd line, where the datamap is extracted (using your own content object).

6.4 Fetching other user information

6.4.1 Fetching user roles

Fetching user roles is straightforward when we have a user node or if we can get hold of the content object id. The following examples should give you what you need to pull out the roles.(adjust this using the previous code examples if you want to extract from an email or a eZContentObject instead).

```
// by username (replace the next line with code from step 2 if you want to fetch by email
address):
```

```
$user = eZUser::fetchByName( 'admin' );
$roles = $user->attribute( 'roles' ); //extract the role IDs
// fetch each one in turn and print:
print_r( $roles );


// if we have an eZContentObject ID (we can use this code if we have a eZContentObject or
an eZContentObjectTreeNode):
$roles = eZRole::fetchByUser( array( 14 ), true );
print_r( $roles );
```

6.4.2 *Fetching user generated content*

Since we have the ID for the user, we can also pull out what content they have created. The eZContentObjectVersion class has the method we need, all we need to do is tell it what status the content should have. In this example we are using just published content but your other options are also given below.

```
$userContent = eZContentObjectVersion::fetchForUser( $users['name']-
>attribute( "contentobject_id" ), eZContentObjectVersion::STATUS_PUBLISHED );
print_r( $userContent );
```

Here are a list of all possible eZContentObjectVersions:

- eZContentObjectVersion::STATUS_DRAFT
- eZContentObjectVersion::STATUS_PENDING
- eZContentObjectVersion::STATUS_PUBLISHED
- eZContentObjectVersion::>STATUS_REJECTED

7 Step 4: Fetching multiple users

The same fetches you can use in your template files can also be used in your PHP scripts. You can also pull out all users and all users of a particular user group or those users that are currently logged in (amongst others). Since there are so many options for pulling out multiple users we will concentrate on the fetch methods in the next tutorial. For the remainder of this tutorial we will cover the other ways you can extract users from eZ Publish.

Please note that for any large website due to the amount of data being extracted if you are running the script through a template or through your web browser there is a very good chance the script will time out so care is needed when running these scripts. One solution is to run the scripts through a cronjob and allow the duration of the script to be extended, that way you can limit the script's use to when your server is at it's quietest.

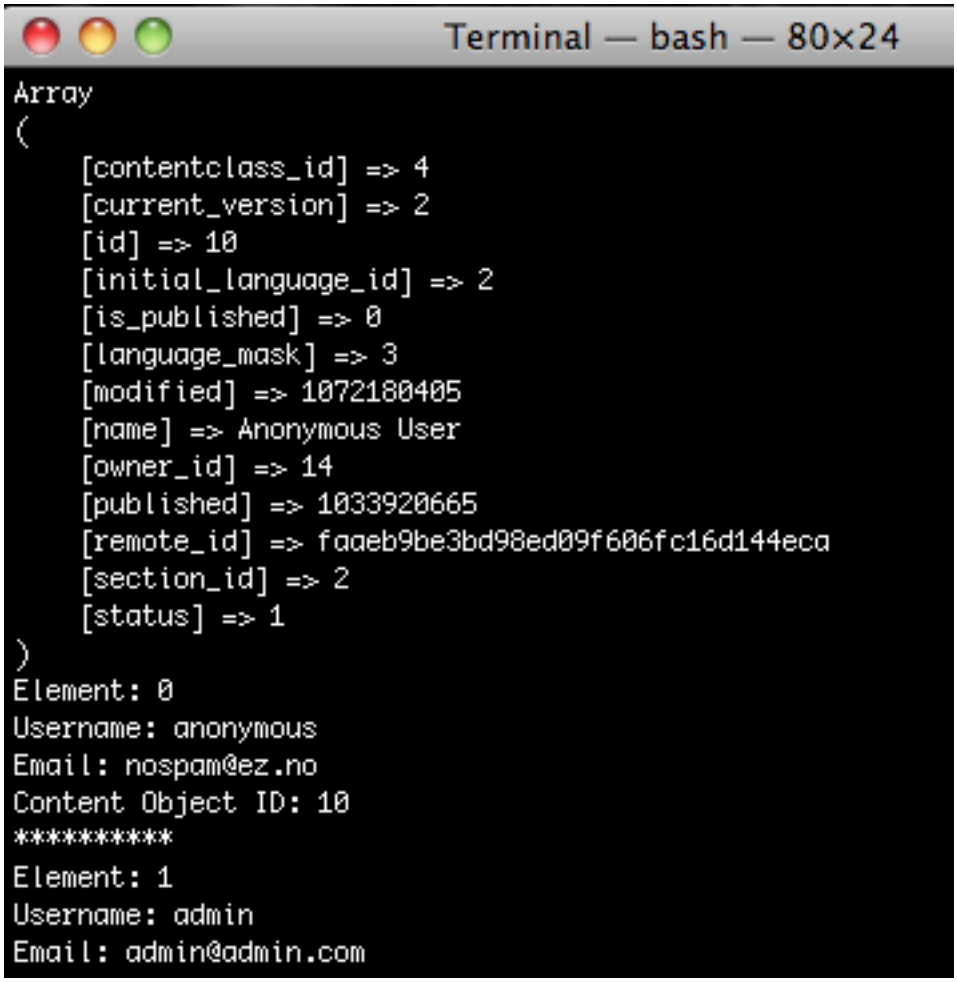
7.1 *Extracting all users*

eZUser has a function to pull out all users currently enabled in your CMS. The code below pulls all of the users out and returns their name and email. You can use the code we have already looked at if you want to pull out other information.

```
$allUsers = eZUser::fetchContentList();
```

```
print_r( $allUsers[0] ); //demonstrating how the data is returned
foreach( $allUsers as $key => $user )
{
       $userObj = eZUser::fetch( $user['id'] );
       $cli->output( 'Element: ' . $key );
       $cli->output( 'Username: ' . $userObj->attribute('login') );
       $cli->output( 'Email: ' . $userObj->attribute('email') );
       $cli->output( 'Content Object ID: ' . $userObj->attribute( 'contentobject_id' ) );
       $cli->output( '**********');
}
```

You may notice a slight change in how we need to pull the users out through this example. The fetchContentList returns an array. This contains only a limited amount of information so then we need to lookup the user using a basic fetch function. We can then extract the information using the code we have already covered. Our example prints the first user object to demonstrate the difference in structure returned by fetchContentList :



7.2 Users by user role

A similar requirement to pulling out all users is to pull out all users for a specifi c role. So for instance, all users of the site will probably have the same role whereas admin users would not. To do this, we need to use the eZRole class. There is a function in the eZRole class that will return all users and user groups for a particular role. If we use this we can then display all users directly, or in the case of user groups we can then access their children, which will all be users.

As roles are usually assigned by group rather than individual users and so most of the time you will be dealing with User Groups. When we have the information we just display the name but you can use the code we've covered previously if you want to show more information :

```
$adminRole = eZRole::fetchByName( 'Administrator' );

$roleUsers = $adminRole->fetchUserByRole(); //this will return eZContentObjects within
separate arrays


foreach( $roleUsers as $key => $userHolder )

{

       $object_type = $userHolder['user_object']->attribute( 'class_name' );

       if ( $object_type == 'user_group' ) //we will more than likely be dealing with a
user group so we need to pull out the users from this

       {

              $user_group = $userHolder['user_object']->attribute( 'main_node' );
//convert so we can access the children of the node


              foreach( $user_group->attribute( 'children' ) as $group_user ) //user_group
will contain  eZContentObjectTreeNodes, let's just output the name (details above on
accessing other fields).

              {

                     $cli->output( 'name: ' . $group_user->attribute( 'name' ) );

              }

       }

       else //if we have a user we are looking at a eZContentObject

       {

              $cli->output( 'name: ' . $userHolder['user_object']->attribute('name') );

       }

}
```

The key part to the script occurs once you;ve established whether you're looking at a user or a user group. If you have a user, it is easy enough to show information almost directly since we are dealing with a eZContentObject (you can use the code from previous examples to display the user information).

For User Groups, we need to carry out additional steps. You will be aware that eZContentObjects do not have a hierarchy in eZ Publish, the hierarchy is established by nodes rather than objects. Therefore, to work out the users which sit under a particular user group (in other words the users who belong to the user group), we must first convert the eZUserObject to a node. The following code in the previous example performs this transformation and then iterates through the users below the group, displaying the name for each user :

```
…
$user_group = $userHolder['user_object']->attribute( 'main_node' ); //convert so we can
access the children of the node


      foreach( $user_group->attribute( 'children' ) as $group_user ) //user_group will
contain  eZContentObjectTreeNodes, let's just output the name (details above on accessing
other fields).
      {
            $cli->output( 'name: ' . $group_user->attribute( 'name' ) );
      }
…
```

7.3 Users currently logged in

The eZUser class also provides methods for providing a list of users who are currently logged in. It is very straightforward to use. The method returns an array of eZUser objects so you you can use the examples in the rest of this tutorial to extract more information.

```
$loggedInCount = eZUser::fetchLoggedInCount();
$loggedIn = eZUser::fetchLoggedInList( true );


$cli->output( 'Logged in Users: ' . $loggedInCount );


foreach( $loggedIn as $user )
{
      $cli->output( 'name: ' . $user->attribute( 'login' ) );
}
```

8 Conclusion

And that's it for part one. There are lots of possible ways to extract the user information from eZ and here we've covered the most basic. We have also seen how we can display all of the user information for the users. The same methods we have used here such as attribute(), definition() and dataMap() are all also applicable to content objects so be sure to make a note of them for other exports you may need to carry out.

The next part of this tutorial we will build on what we've started by formatting and displaying filtered lists of user information and setting up a cronjob that can be used to automate the process, saving the results in tab delimited files for easy export.

9 Resources

- http://serwatka.net/blog/fetching_ez_publish_content_objects_with_php - Fetching eZ Publish content objects with PHP
- 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

10 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.

11 License choice