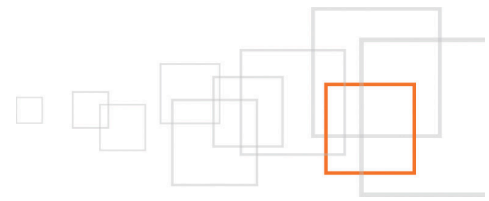


Creating eZ Publish Objects in PHP

By David Linnard

<http://www.onequarterenglish.co.uk>



Index

1	Goal description	3
2	Introduction	3
3	Pre-requisites and target population	3
4	Step 1 – Setup the script	3
5	Step 2 - Who will be importing?	4
5.1	Current User	4
5.2	A Specific User.....	4
5.2.1	Specific User by Name	5
5.2.2	Specific User by Email.....	5
6	Step 3 - Adding the Content.....	5
6.1	Simple Content – A Folder.....	5
6.1.1	Retrieving background information.....	7
6.1.2	Preparing general object information	7
6.1.3	Preparing the object attributes.....	8
6.1.4	Creating the object	8
6.2	Importing XML Fields	11
6.3	Importing Images and Files	12
6.4	Adding related objects.....	14
6.4.1	Adding related objects to an attribute.....	14
6.4.2	Adding related objects to an object	15
6.4.3	Adding Other Content.....	15
7	Conclusion	15
8	Resources	15
9	About the author : David Linnard	16
10	License choice	16

1 Goal description

Creating eZ Publish objects using PHP to allow you to dynamically create or import content from elsewhere. After reading this article you should be able to create objects for any situation including those with XML fields, image or file fields and object relations.

2 Introduction

eZ Publish allows you to publish nodes directly through your PHP very easily. In this tutorial I will cover how you can write a custom script to import your content into the CMS. Importing images, XML content and object relations along the way. The code be easily adapted to suit other purposes, whether you want to add it to your own extension to handle bulk imports, or if you want to automate content import for third party content using a cronjob.

3 Pre-requisites and target population

Knowledge of object orientated PHP and the ability to run PHP scripts on the command line is required. A knowledge of the structure of the eZ Publish admin interface is required for establishing the structure of what you will be creating. Being able to access class information under setup is also needed.

Additional knowledge about eZ Content Objects is beneficial as is the structure of eZ Publish URLs.

An installed instance of at least eZ Publish 3.9 is required although I am running the scripts in eZ Publish 4.3.

4 Step 1 – Setup the script

For this tutorial, we are going to stick to running the script from the command line. To do this, we are going to create a script within the directory `/bin/php` which is found in the root of your eZ Publish site. This script is the standard layout for an eZ Publish PHP script. To make debugging easier for us, I have enabled it in the script options. Once you are happy with your code, set the debug options to false.

Code :

```
<?php
set_time_limit ( 0 );
require 'autoload.php';
$cli = eZCLI::instance();
$db = eZDB::instance();
$script = eZScript::instance( array( 'description' => ( "eZ Publish data import.\n\n" .
    "Simple import script for use with eZ Publish"),
    'use-session' => false,
    'use-modules' => true,
    'use-extensions' => true,
    'debug-output' => true,
    'debug-message' =>true) );
```

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

/*****
Our Functionality will go here
*****/

$script->shutdown();
?>
```

5 Step 2 - Who will be importing?

Before you import any data you need to work out which eZ Publish user will be carrying it out? If you are using a template which is calling this functionality then you can pull out the current user. If you are running the script from the command line, you do not have a current user and so you may want to choose or create a specific one

For the rest of the steps of this tutorial, replace the following code with that you want to use for your import (starting with this step).

Code :

```
/*****
Our Functionality will go here
*****/
```

5.1 *Current User*

Pulling in the current user is easy enough, using the static method `currentUser()` from within `eZUser` class you can pull out the user currently logged in. If nobody is logged in, the anonymous account within the CMS is used (typically if you run the function in a custom PHP Script you will be returned the anonymous user. I would recommend using the current user if the import has been triggered by an action from the user on the site.

Code :

```
$user = eZUser::currentUser();
```

5.2 *A Specific User*

You will typically want to do this if you are running a PHP script and I would recommend creating an import user for the task. You can pull out a specific user by name or email. Alternatively you can statically use an ID for the creation process but you may find these vary across your dev and live environments so I wouldn't recommend it.

5.2.1 Specific User by Name

Code :

```
$user = eZUser::fetchByName( 'Import' );  
if (!$user){//if no user exists let's pull out the current user:  
    $user = eZUser::currentUser();  
}
```

5.2.2 Specific User by Email

Code :

```
$user = eZUser::fetchByEmail( 'Import@admin.com' );  
if (!$user){//again, default to current user if necessary  
    $user = eZUser::currentUser();  
}
```

6 Step 3 - Adding the Content

We'll now start importing our content. Although the basic process is straightforward, there are a couple of things to look out for so we'll look at importing different content in turn. eZ Publish makes use of the fromString methods when carrying out the import. If the object you are creating contains fields of types we are not covering here, I'd recommend checking out the fromString documentation found here.

For the examples below we will not be using any hard coded node ids. These result in massive complications when you are moving your files between your development, staging and live environments and so I would advise avoiding them whenever possible. It also makes the code much simpler to understand and easier to follow when you do not use them.

6.1 Simple Content – A Folder

We'll start off by importing the most simple thing we can: a folder. We will ignore the description for now and concentrate on getting the object into eZ Publish. I've created a folder called "My Imported Stuff" in the Content Structure Home folder of the CMS so we will import content into there. Don't worry about the complexity of it. We will break it down after we have used the script.

Code :

```
//getting information required to setup the node:  
$user = eZUser::fetchByName( 'Import' );//import our user (replace with the code for the  
previous step if necessary)  
if (!$user){//if no user exists let's pull out the current user:  
    $user = eZUser::currentUser();  
}  
$cli->output( 'Username: '.$user->Login );
```

```

$parent_node = eZContentObjectTreeNode::fetchByURLPath('my_imported_stuff');
$cli->output('Parent: '.$parent_node->Name);

/*We will use this to tell eZ where our new node will be stored, note the underscores
rather than hyphens and that it is all in lowercase.*/

//setting general node details
$params = array();
$params ['class_identifier'] = 'folder'; //class name (found within setup=>classes in the
admin if you need it
$params['creator_id'] = $user->ContentObjectID;//using the user created above
$params['parent_node_id']=$parent_node->NodeID;//pulling the node id out of the parent
$params['section_id'] = $parent_node->ContentObject->SectionID;
/*we don't need to do this as the section will default to that of the parent but if you
want to use a different node for the section the same approach can be used, just pull out
the node with the section you are using and then pull the SectionID from it.*/

//setting attribute values
$attributesData = array ( ) ;
$attributesData['name'] = 'A brand new folder' ;
$attributesData ['short_name'] = 'Shorter name' ;
$params['attributes'] = $attributesData;

print_r($params);//lets print out the data so we know exactly what is being stored

//publishing the content:
$contentObject = eZContentFunctions::createAndPublishObject($params);

if ($contentObject)
{
    $cli->output('=====');
    $cli->output('Output:');
    $cli->output("Content Object ID: ".$contentObject->ID);
    $cli->output("Name: ".$contentObject->Name());
    $cli->output("Data Map: ".print_r($contentObject->DataMap(),true));
}

```

You should now have a script you can use! Make sure you replace the user details you are using and also replace the name of the folder you are saving into if necessary. We can split this script up into four parts:

6.1.1 Retrieving background information

Code :

```
//getting information required to setup the node:
$user = eZUser::fetchByName( 'Import' );//import our user (replace with the code for the
previous step if necessary)
if (!$user){//if no user exists let's pull out the current user:
    $user = eZUser::currentUser();
}
$cli->output('Username: '.$user->Login);

$parent_node = eZContentObjectTreeNode::fetchByURLPath('my_imported_stuff');
$cli->output('Parent: '.$parent_node->Name);

/*We will use this to tell eZ where our new node will be stored, note the underscores
rather than hyphens and that it is all in lowercase.*/
```

The first section of our script just works out who should be importing the content and where they should be importing it to. We've covered the user code but the next line is equally important as it extracts the node details for the node we want our new folder to sit under:

Code :

```
$parent_node = eZContentObjectTreeNode::fetchByURLPath('my_imported_stuff');
```

The `fetchByURLPath` is perfect for this, there are a couple of things to be aware of with the function:

- The `fetchByURLPath` uses underscores instead of hyphens to replace characters such as whitespace
- The path is all in lower case
- To retrieve a node from a different directory separate the directories with a forward slash (as you would expect). For example:

Code :

```
$parent_node = eZContentObjectTreeNode::fetchByURLPath('media/images');
```

6.1.2 Preparing general object information

Code :

```
//setting general node details
$params = array();
$params ['class_identifier'] = 'folder'; //class name (found within setup=>classes in the
admin if you need it
$params['creator_id'] = $user->ContentObjectID;//using the user created above
$params['parent_node_id']=$parent_node->NodeID;//pulling the node id out of the parent
$params['section_id'] = $parent_node->ContentObject->SectionID;
/*we don't need to do this as the section will default to that of the parent but if you
```

want to use a different node for the section the same approach can be used, just pull out the node with the section you are using and then pull the SectionID from it.*/

In this section we start storing the information eZ Publish requires to create the node. We will pass eZ Publish the \$params array when we tell it to create the folder. First we tell it the type of object we will be creating and then we pull out the user id and the node id that we established in the first part of the script.

The section_id is stored only for completeness and your example will work fine without it. If you need your node to be off a different section to it's parent, you can either use the eZSection::fetchFilteredList method to pull out the section by name, or if you want to use a section of a specific node, use eZContentObjectTreeNode::fetchByURLPath as we have done above and use the path to that node. The section_id can then be extracted using the same code as we have used above.

6.1.3 Preparing the object attributes

Code :

```
//setting attribute values
$attributesData = array ( ) ;
$attributesData['name'] = 'A brand new folder' ;
$attributesData ['short_name'] = 'Shorter name' ;
$params['attributes'] = $attributesData;

print_r($params); //lets print out the data so we know exactly what is being stored
```

This is very straightforward, we just store the name of each attribute along with it's value as a set of key/value pairs. Once we have done this we add it to our \$params array ready for the final step...

6.1.4 Creating the object

Code :

```
//publishing the content:
$contentObject = eZContentFunctions::createAndPublishObject($params);

if ($contentObject)
{
    $cli->output('=====');
    $cli->output('Output:');
    $cli->output("Content Object ID: ".$contentObject->ID);
    $cli->output("Name: ".$contentObject->Name());
    $cli->output("Data Map: ".print_r($contentObject->DataMap(), true));
}
}
```

And that is it! We have already stored all of the values that eZ Publish needs so all we need to do to publish it is send the eZContentFunctions::createAndPublishObject function the \$params array. The function returns an eZContentObject which we can then use if needed. We are simply outputting it's ID, name and the data it

is storing.

Below is the output to our script. From it you can see the structure of the \$params array we use to populate the object quite clearly. You can also see how easy it then is to extract the information we have just published.

```
Terminal — bash — 118x59
unknown-00-25-00-f9-a9-5c:ezPublications d_linnard$ php bin/php/import_content.php
Username: Import
Parent: Images
Array
(
    [class_identifier] => folder
    [creator_id] => 90
    [parent_node_id] => 51
    [section_id] => 3
    [attributes] => Array
        (
            [name] => A brand new folder
            [short_name] => Shorter name
        )
)
)
=====
Output:
Name: 101
Name: Shorter name
Data Map: Array
(
    [name] => ezContentObjectAttribute Object
        (
            [HTTPValue] =>
            [Content] =>
            [DisplayInfo] =>
            [IsValid] =>
            [ContentClassAttributeID] => 4
            [ValidationError] =>
            [ValidationLog] =>
            [ContentClassAttributeIdentifier] => name
            [ContentClassAttributeCanTranslate] =>
            [ContentClassAttributeName] =>
            [ContentClassAttributeIsInformationCollector] =>
            [ContentClassAttributeIsRequired] =>
            [PersistentDataDirty] =>
            [InputParameters] =>
            [HasValidationError] =>
            [DataTypeCustom] =>
            [ID] => 468
            [ContentObjectID] => 101
            [Version] => 1
            [LanguageCode] => eng-GB
            [LanguageID] => 2
            [AttributeOriginalID] => 0
            [SortKeyInt] => 0
            [SortKeyString] => a brand new folder
            [DataTypeString] => ezstring
            [DataText] => A brand new folder
            [DataInt] =>
```

Although that is a simple example it is the basis for all of the following code. To test it has worked go into the back office and verify the node has been created. The basic script is simple enough but there are a few caveats you need to be aware of. Images, files, XML and related object fields can all be added using this approach but they are slightly more complicated to import, we'll start with XML fields and then we will work through the others:

6.2 Importing XML Fields

XML fields are more difficult due to the structure they are stored in within the database. You need to make sure you wrap your text within the extra xml layers to add it successfully. The folder object also has a description field we omitted in the first example so let's look at the code needed to add content to that. eZ Publish has some built in methods for converting html to the correct format so we can just use these.

This code should be used in the step "Preparing the object attributes" found above:

Code :

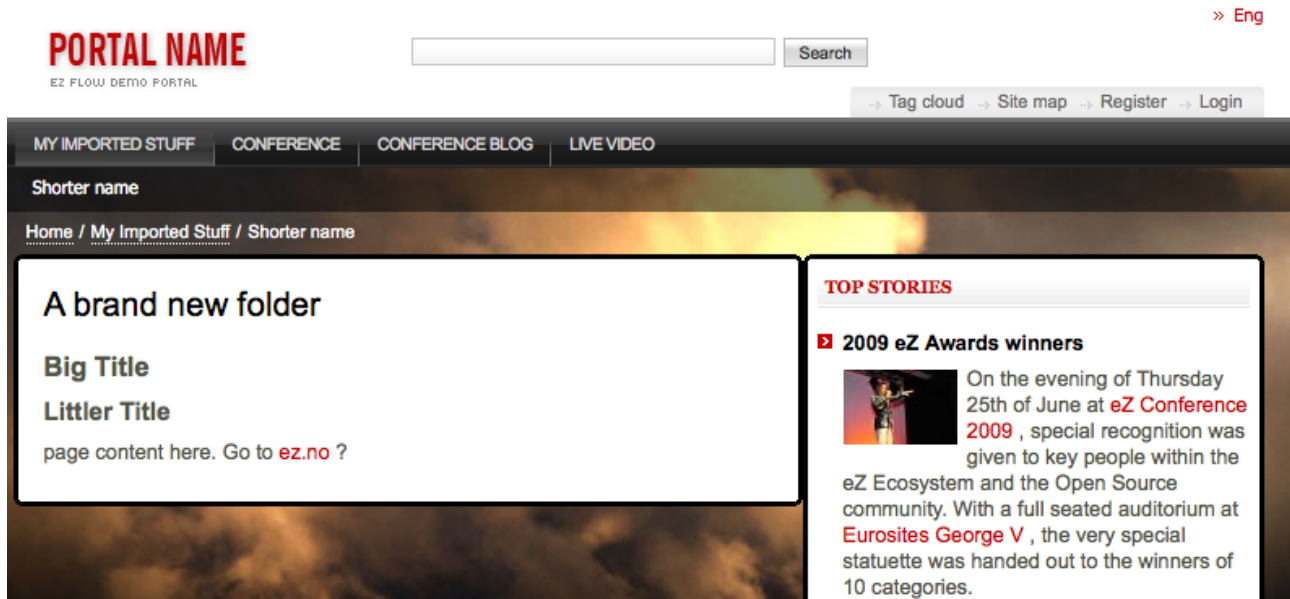
```
...
//setting attribute values
...
//updating the structure for the valid XML
$xmlContent = '<h1>Big Title</h1><h2>Littler Title</h2><p>page content here. Go to <a href="http://www.ez.no">ez.no</a>?</p>'; //my example content
//creating and setting up the parser
$parser = new eZSimplifiedXMLInputParser( );
$parser->setParseLineBreaks( true );
//parsing the content
$document = $parser->process( $xmlContent );

//adding the content to an object
$attributesData ['description'] = eZXMLTextType::domString( $document );
...
$contentObject = eZContentFunctions::createAndPublishObject($params);
```

Below is an example of this XML content before and after it is passed into the XML parser. As you can see a lot of extra information is added to the HTML which would be difficult to add manually.

```
Original Content:
<h1>Big Title</h1><h2>Littler Title</h2><p>page content here. Go to <a href="http://www.ez.no">ez.no</a>?</p>
=====
XML Based Version:
<?xml version="1.0" encoding="utf-8"?>
<section xmlns:image="http://ez.no/namespaces/ezpublish3/image/" xmlns:xhtml="http://ez.no/namespaces/ezpublish3/xhtml/" xmlns:custom="http://ez.no/namespaces/ezpublish3/custom/"><section><header>Big Title</header><section><header>Littler Title</header><paragraph>page content here. Go to <link url_id="33">ez.no</link>?</paragraph></section></section>
```

Whenever this is displayed to the user, the formatting will be converted back to HTML. If we add the XML code to our previous example, the result is as follows:



As you can see, the formatting has been reverted back to HTML. Please note that not all HTML tags can be imported into XML (although custom tags can be added when necessary), for a complete list of the tags available by default, please see the documentation.

6.3 Importing Images and Files

Images and files are added by using in the name of your file so they quite straightforward. The only addition is that you need to make sure to specify the storage directory for the file so that eZ knows where to look. If you have issues with a missing image then the problem is most likely going to be there (if you are running the script from the command line make sure your debug is turned on). The following code is based on the previous example. Again, check the CMS admin panel to make sure the image has been created:

```
//getting information required to setup the node:
$user = eZUser::fetchByName( 'Import' );//import our user (replace with the code for the
previous step if necessary)
if (!$user){//if no user exists let's pull out the current user:
    $user = eZUser::currentUser();
}
$cli->output('Username: '.$user->Login);

$parent_node = eZContentObjectTreeNode::fetchByURLPath('media/images/imported_images');
$cli->output('Parent: '.$parent_node->Name);

//setting node details
$params = array();
$params ['class_identifier'] = 'image';
```

```

$params['creator_id'] = $user->ContentObjectID;//using the user extracted above
$params['parent_node_id']=$parent_node->NodeID;//pulling the node id out of the parent
$params ['storage_dir' ] = $_SERVER['PWD'].' /var/ezflow_site/storage/import_images/';
/*required so ez knows where to look. The ending "/" required. $_SERVER['pwd'] is being
used as the script is being run through the command line. I've created the folder
"import_images" on the server and moved my image into it.*/

//setting attribute values
$attributesData = array ( ) ;
$attributesData['name'] = 'Adding a random image' ;
$attributesData [ 'image' ] = 'my_image.jpg' ;

//storing xml content for the caption
$xmlContent = "<p>David's Picture Test</p>";
$parser = new eZSimplifiedXMLInputParser( );
$parser->setParseLineBreaks( true );
$document = $parser->process( $xmlContent );
$xmlString = eZXMLTextType::domString( $document );
$attributesData ['caption'] = $xmlString;

$params['attributes'] = $attributesData;

//publishing node
$imageObject = eZContentFunctions::createAndPublishObject($params);

```

Most of this should be familiar to before but these are the lines to look out for:

Code :

```

$params ['storage_dir' ] = $_SERVER['PWD'].' /var/ezflow_site/storage/import_images/';

```

This tells eZ where to look, we are running the script from the command line, otherwise you would need to change \$_SERVER['PWD']. Also note the trailing "/" which is necessary. In this instance, I've created a folder called import_images which is stored within the var directory.

Code :

```

$attributesData [ 'image' ] = 'my_image.jpg' ;

```


And this is our image. We are also storing a description on this object which utilises the XML information we used previously. Once we run the script we should now have a new image accessible through the CMS:

You are here: [Media](#) / [Images](#) / [Imported Images](#) / Adding a random image


Media library

- Media
- Trash

 **Adding a random image [Image]**

Last modified: 20/05/2010 10:25 pm, [Import User](#) (Node ID: 105, Object ID: 103) English (United Kingdom) 

Preview Details Locations (1) Relations (0)



Edit Move Remove Manage versions

6.4 Adding related objects

6.4.1 Adding related objects to an attribute

Various types in eZ Publish uses a related object as an attribute (so that you can specify a file which can be attached to an article, for instance). To do this you just need to supply the object ID you just created as the attribute. The code below assumes you have just created an image using the code above. We are then going to use the object id of the image to attach the image to an image gallery.

Code :

```
//after the previous example store the object ID of the image:
$image_id = $imageObject->ID;

...
//then, when you specify the attributes of the image gallery use the ID:
$attributesData ['image'] = $image_id;
```

Although this approach will work in certain instances, there are other times it will not. For instance, The example of Image Galleries we have just used. The problem is that the image we will want to add to the gallery should be sitting below it. Since we have not even published the gallery yet we have no way at this point of attaching something which should sit underneath it.

The solution for this is to publish without the related object initially. There is a comparable method to the one we have been using in this tutorial for updating content and so when we have added all of the images as well, we can then call the update method instead. The update method is a static method called `eZContentFunctions ::updateAndPublishObject`. If you bring up the class definition within your IDE then there is an example of it in use (the file is located here: `kernel/classes/ezcontentfunctions.php`).

6.4.2 *Adding related objects to an object*

Since every node in eZ Publish can also have other related objects this is also a key thing you may have to do. It is not strictly possible with the `createAndPublishObject` function we are using but the function returns the content object that has been created. Due to this we can attach the related objects directly to this. The code below assumes we are pulling out the image id as we were previously and attach it to another node:

Code :

```
$image_id = $imageObject ->ID;//first we need the object id, let's pull out the image ID
as we did before

...

//create & publish the object we need the image to link to

...

$contentObject-
>appendInputRelationList(array($image_id),eZContentObject::RELATION_COMMON);

//create the relation

$contentObject->commitInputRelations($contentObject->CurrentVersion);//commit to store it
in the database
```

6.4.3 *Adding Other Content*

If you want to extend the script for further variable types then I would check out the `FromString` method documentation. This covers examples such as Date, matrices and keywords.

7 Conclusion

You should now be quite comfortable with creating objects in eZ Publish and it should hopefully be simple enough to extend the basic script you have for your specific purpose. The code can be used wherever you need it. In particular, I would suggest it's use in implementing cronjobs to automate external content import (please also be aware of the built in RSS feed import under the settings tab for simple RSS importing) and also using it in extensions for bulk producing objects (for instance I have previously used a similar script to import a galleries worth of images at a time into the system).

The `fromstring` and `createAndPublish` methods have made importing data into eZ Publish very straightforward. If you found this useful I would also suggest checking out the `update` method which exists in the `eZContentFunctions` class (The file location in eZ Publish is: `kernel/classes/ezcontentfunctions.php`).

8 Resources

- Forum Post on creating objects - <http://share.ez.no/forums/developer/right-way-to-create-an-ez-object-using-php>
- Command Line Scripts - http://ezpedia.org/ez/command_line_scripts
- FromString Method - http://ezpedia.org/en/ez/simple_fromstring_and_tostring_interface_for_attributes
- XML Tags Documentation - http://ez.no/doc/ez_publish/technical_manual/4_x/reference/xml_tags
- eZContentFunctions class reference - http://pubsvn.ez.no/doxygen/4.0/html/ezcontentfunctions_8php-source.html

9 About the author : David Linnard



David is a London based web developer with a wide variety of skills who has spent the last two years on commercial eZ Publish web builds. He is also experienced at handling a variety of other content management systems and the Zend Framework.

10 License choice

Available under the Creative Commons Attribution Non-Commercial License