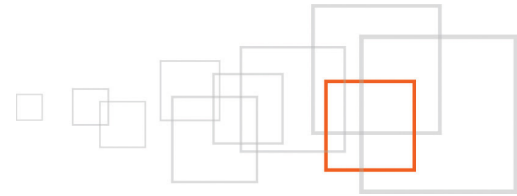


# Creating a simple custom workflow event

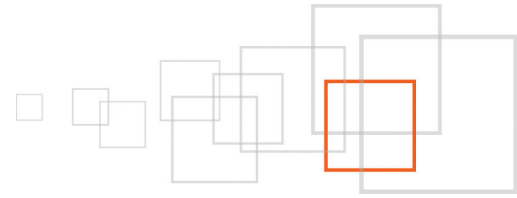
---

**By Quoc-Huy NGUYEN DINH**  
<http://www.qhphotography.com/>



## Index

1	Goal description.....	3
2	Introduction.....	3
3	Pre-requisites and target population.....	3
4	Step 1 – What is a workflow?.....	4
5	Step 2 – Setting up a default workflow for content approval.....	4
6	Step 3 – Develop a workflow event: preparation.....	10
7	Step 4 - Develop a workflow event: Arcgo OAuth Twitter library.....	11
8	Step 5 - Develop a workflow event: content class attribute.....	14
9	Step 6 - Develop a workflow event: posting a Twitter status.....	15
11	Step 7 - Develop a workflow event: setup the workflow.....	17
12	<u>Step 8 – Authorize eZ Publish via Twitter OAuth.....</u>	<u>18</u>
13	Conclusion.....	24
14	Resources.....	24
15	Credits.....	24
16	About the author, Quoc Huy Nguyen Dinh.....	25
17	License choice.....	25



## 1 Goal description

This tutorial will guide you through the development of a simple eZ Publish workflow event. At the end of the tutorial, you should be able to create your own workflow event, configure a workflow that would execute the event and configure workflow triggers in the admin interface.

## 2 Introduction

eZ Publish comes with several default workflow event: approval, payment gateway etc... In some occasions you will probably need to create your own one. Workflow events are very useful in the case where you need some actions to be done on a published object either before the actual publication occurs (maybe a kind of auto-validation of some content...) or after (generation of a JPEG image from a text field, post publishing notification system...).

In this tutorial we will see how to create a simple workflow event that sends a post to Twitter with a shorten URL (TinyURL) of the node being published.

## 3 Pre-requisites and target population

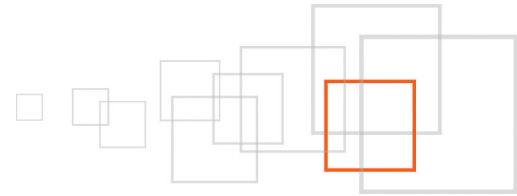
eZ Publish templating language is quite powerful and for a lot of work it's all you need. But for this type of development a knowledge of object oriented PHP is required.

Along with how to develop your own workflow event, this tutorial will also explain how to setup workflows and triggers so no previous knowledge is required.

Also require is an understanding of eZ Publish's structure, of the use of INI files and of the creation/editing of content classes.

As an example application for this tutorial we will be using a quite trendy subject: Twitter. So you will need a test Twitter account and obviously a working test eZ Publish installation from version 4.0+ (it should be working for previous versions, not tested though).

Initially this tutorial was supposed to target intermediate eZ Publish developers, the fact that Twitter has changed their authentication system has made it a bit more complexe. So as a bonus this tutorial will also briefly cover the use of a CLI script using the eZScript class. This will require you to have a good knowledge of command line scripts and a terminal access (ssh) to your server as the Twitter authentication will be done through it.



## 4 Step 1 – What is a workflow?

There are two good descriptions of eZ Publish workflows here:

<http://doc.ez.no/eZ-Publish/Technical-manual/4.x/Concepts-and-basics/Workflows>

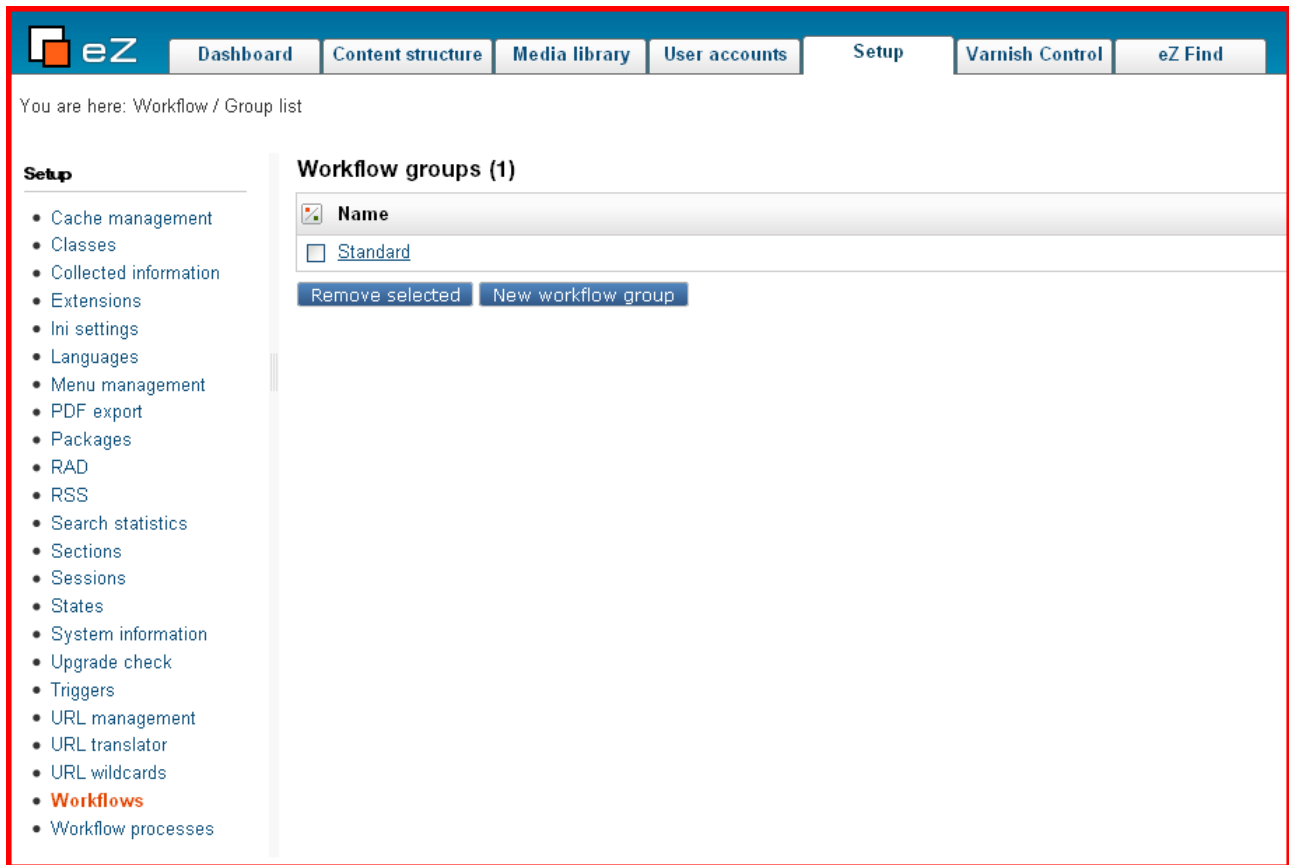
<http://www.martinbauer.com/eZ-publish-Overview/Workflow>

But in brief, eZ Publish workflow is a “system that allows you to create a process of functions with or without user intervention”. eZ Publish *workflows* consists of one or multiple *events* that can be organized into *workflow groups*. A *workflow* is initiated by a *trigger*. Workflows are very useful when you need to have something done when a node has been newly created or published: content approval process, auto-validation, auto-filtering of forbidden words, notification to social media services, you name it.

## 5 Step 2 – Setting up a default workflow for content approval

To get familiar with eZ Publish workflows, let setup the default workflow event “Event/Approve” that to have comments approved (we are here talking about the old “Comment” class not the new eZ Comment system). This, of course, can be applied to other type of content classes.

First, login into the admin interface and from the top menu click on “Setup” (you’ll need enough privileges to reach this area and some administrators might have disable access to this area from a site.ini config file). This should bring you to the Cache management page. From the left menu click on the menu “Workflows”



The screenshot shows the eZ Publish admin interface. At the top, there is a navigation bar with the eZ logo and several menu items: Dashboard, Content structure, Media library, User accounts, Setup, Varnish Control, and eZ Find. Below the navigation bar, the breadcrumb path reads "You are here: Workflow / Group list". On the left side, there is a "Setup" menu with a list of options: Cache management, Classes, Collected information, Extensions, Ini settings, Languages, Menu management, PDF export, Packages, RAD, RSS, Search statistics, Sections, Sessions, States, System information, Upgrade check, Triggers, URL management, URL translator, URL wildcards, Workflows (highlighted in orange), and Workflow processes. The main content area is titled "Workflow groups (1)" and contains a table with one row: "Standard". Below the table, there are two buttons: "Remove selected" and "New workflow group".

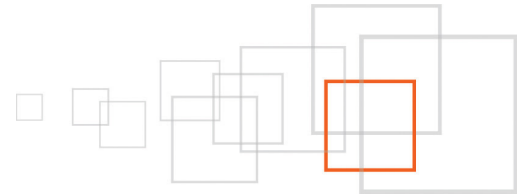


Fig. 1

As you can see from the above screenshot (Fig. 1), you are now in a page displaying a list of “Workflow groups”. By default there is only one group named “Standard”. Click on “Standard” to browse its content. As per Fig.2, below, there is no workflow created in the “Standard” workflow group by default.

Lets create a new one, it will execute the workflow event for approving content. By its nature, a content approval needs to occur before publication. So we will call our work flow “Before publishing”. To create a new workflow, click on the button “New workflow” (bottom right). This will bring you to the edit page, see Fig. 3 below. For the name, type in “Before publishing”. Within this page, you are able to assign an event to this workflow by selecting and add it using the drop down menu at the bottom. Lets add it and see why this is not useful for us in this situation, select “Event / Approve” from the drop down menu and then click on “Add event” button. Fig. 4 shows you what you should see.

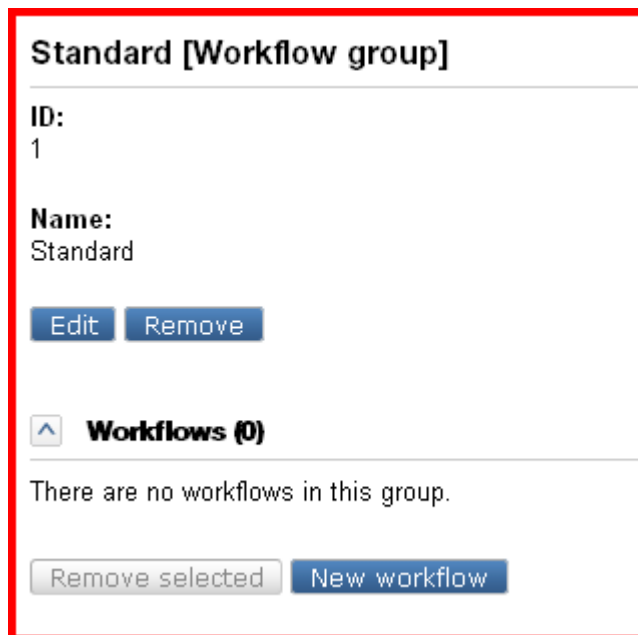


Fig. 2

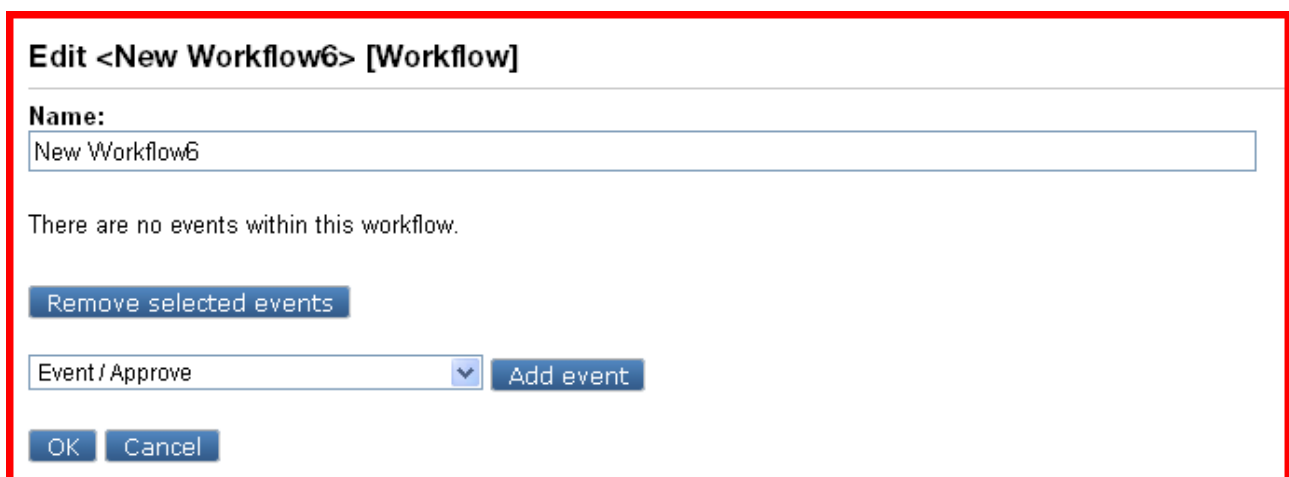
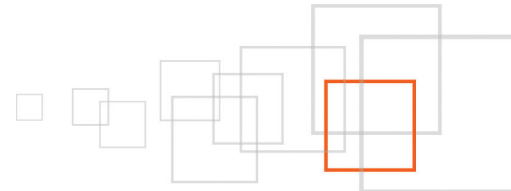


Fig. 3



### Edit <Before Publishing> [Workflow]

**Name:**

1(1) Event / Approve ↓ ↑

**Description / comments:**

**Affected sections:**  
  
Blog  
Design  
Hidden  
Lighting Diagrams

**Affected languages:**  
  
English (United Kingdom)  
French (France)

**Affected versions:**  
  
Publishing new object  
Updating existing object

**Users who approve content**  
No users selected.

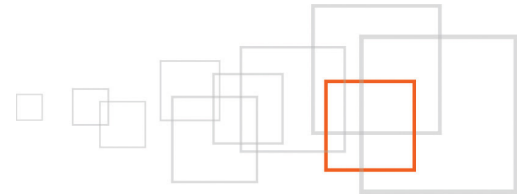
**Groups who approve content**  
No groups selected.

**Excluded user groups ( users in these groups do not need to have their content approved )**  
No groups selected.

Fig. 4

The field "Description / comments" is optional and is for clarity, you can describe what this event is doing. The "Event / Approve" event comes up with few filters that allow you to define when this trigger should be executed:

- **Affected sections:** the event will be executed when an object has been published and is part of a node that belongs to the eZ Publish sections selected
- **Affected languages:** same here but applies to object published in a certain language
- **Affected versions:** do you want to execute the event for a newly created object, on update to an existing object or in both cases?



The next sections allow you to select users or groups of users allowed to approve pending items and groups of users that do not need to have their content approved.

Adding this event directly to our new workflow does not allow us to apply the content approval event to only Comments, this will be applied to any content being published! To achieve this requirement we will first remove the event we just added, create another workflow and come back to this one and configure a workflow multiplexer.

First, lets delete this event: tick the box next to "1(1) Event / Approve" then click on "Remove selected events" at the bottom. Then click on "OK" at the bottom of the page to save our "Before publishing" workflow.

Back to the workflow list of the "Standard" workflow group, create a new workflow and name it "Content approval". Add the "Event / Approve" event and configure it for "All sections"; "All languages"; "All versions". During the development and test we will add our self in the "Users who approve content" list. Alternatively you could also add your group into "Groups who approve content". Now click on "OK".

Back to the workflow list of the "Standard" workflow group, you should now see two workflows: "Before publishing" and "Content approval". The latter is applying content approval process to any content being published. But lets edit "Before publishing"; by either clicking on it and then on "Edit" or by clicking on the pencil icon on the right. In this workflow instead of adding a "Event / Approve" event, lets add "Event / Multiplexer". The multiplexer event will allow you to run another workflow within our current workflow but with more control option (see Fig. 5).

**Edit <Before Publishing> [Workflow]**

**Name:**  
Before Publishing

1(1) Event / Multiplexer

**Description / comments:**  
Comment approval

**Affected sections:**  
All sections  
Blog  
Design  
Hidden  
Lighting Diagrams

**Affected languages:**  
All languages  
English (United Kingdom)  
French (France)

**Classes to run workflow:**  
Article (sub-page)  
Banner  
Blog  
Blog post  
Comment

**Affected versions:**  
All versions  
Publishing new object  
Updating existing object

**Users without workflow IDs:**  
Users  
Members  
Administrator users  
Editors  
Anonymous Users

**Workflow to run:**  
Content Approval

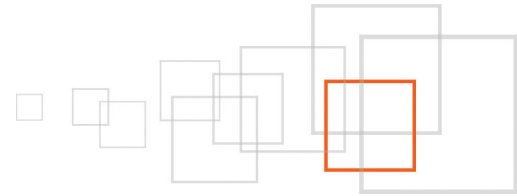
Remove selected events

Event / Approve Add event

OK Cancel

Fig. 5

Now you can see a new filter column called "Classes to run workflow". This will allow us to select "Comment" and run this event only on "Comments" objects being published. The selection "Users without workflow IDs" allows us to select group of users not affected by this multiplexer event. You could unselect everything and configure it in our "Content approval" workflow by adding the groups in the users that don't need approval



selection box.

In the “Workflow to run” drop down box, select our “Content approval” workflow and click on “OK”.

We have now setup a workflow “Before publishing” that is running another workflow “Content approval” using a multiplexer event. This execution will occur only on “Comments” items being published.

Is that it? No! A workflow won't be executed if you don't attach it to a workflow trigger. So lets configure one, click on “Triggers” from the left hand side menu:

**Workflow triggers (9)**

Module	Function	Connection type	Workflow
content	publish	before	Before Publishing <input type="button" value="v"/>
content	publish	after	After publishing <input type="button" value="v"/>
shop	confirmorder	before	No workflow <input type="button" value="v"/>
shop	checkout	before	No workflow <input type="button" value="v"/>
shop	checkout	after	No workflow <input type="button" value="v"/>
shop	addtobasket	before	No workflow <input type="button" value="v"/>
shop	addtobasket	after	No workflow <input type="button" value="v"/>
shop	updatebasket	before	No workflow <input type="button" value="v"/>
shop	updatebasket	after	No workflow <input type="button" value="v"/>

Fig. 6

Fig. 6 shows you the list of triggers available in eZ Publish. Adding a new trigger is out of the scope of this tutorial. What we are interested in this screen is the two triggers from the “content” module (the two first ones). As you can guess, the first one is a trigger that would execute a workflow *before* the *publish* process of a *content*. In the “Workflow” column, select our workflow group “Before publishing” for the “content / publish / before” trigger, then click on “Apply changes”.

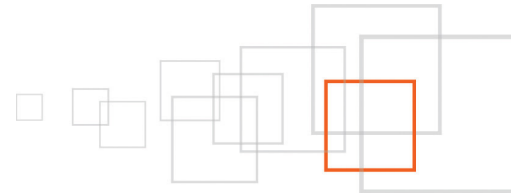
Now, we're done.

Every time a content will be published in eZ Publish, the CMS will trigger the “Before Publishing” workflow before publishing the actual content (so the content is still a draft). In turn, the “Before Publishing” workflow is using a multiplexer that will check few rules before executing the other workflow “Content Approval” that is actually executing the content approval process by running the “Event / Approve” event.

From now on, if a user add a comment, it will be sent out pending for approval. The approvers will find a list of pending items by logging in the admin interface and click on the “Collaboration” link on the Dashboard (for eZ Publish versions prior to 4.3, the “Collaboration” area is found under “My account” from the top menu.

We will not go into details on this “Collaboration” tool, but here's how it looks like in Fig. 7:





## Approval

The content object needs your approval before it can be published.

Do you approve of the content object being published?

**Comment:**

Add Comment

Approve

Deny

## Preview

Published at: 29/09/2010 6:37 am, [Anonymous User](#)

English (United Kingdom) 

Default object view. [Click to create a custom template](#)

## Subject

### Message

great idea.. the pics dont load though for bigger view :(

### Name

Ryan

**Email (so I can respond to your request)**

### Website

<http://>

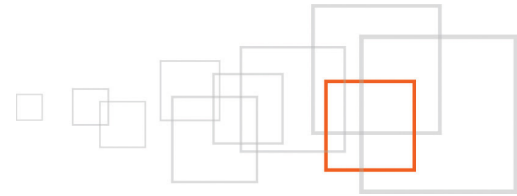
### Placed in

## Participants

**Author:**

**Approver:**

Fig. 7



## 6 Step 3 – Develop a workflow event: preparation

As an example to this tutorial we will see how to create a new workflow event that will automatically post a Twitter status update on publication of a blog post. The status message is customizable and the editor can choose not to post any.

We need to create a new folder in the existing extension/ folder. Lets call it “mytwitter”. Several sub-folders need to be created:

- eventtypes/: this folder will host the actual code of our workflow event.
- lib/: this folder will host 3<sup>rd</sup> party libraries needed by our eZ Publish extension.
- settings/: some settings files.

Lets start with the “eventtypes/” folder. Inside this folder, you need to create another one and name it “event/”. It will hold all the events we want to create as part of this extension. For our extension we will have just one event so lets create a new folder and name it “twitterstatusupdate/”. You should be have  
~extension/mytwitter/eventtypes/event/twitterstatusupdate/

Inside this folder, lets create our PHP file. The filename of our file must be the name of the current folder (twitterstatusupdate) and end with “type.php”. So we will need to create a file named “twitterstatusupdatetype.php”.

We will start by filling it in with an empty template for developing a workflow event:

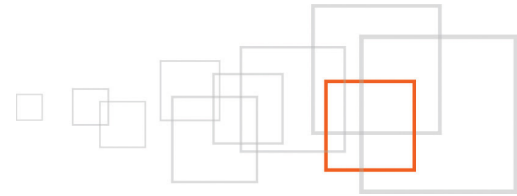
```
<?php

class TwitterStatusUpdateType extends eZWorkflowEventType
{
    const WORKFLOW_TYPE_STRING = "twitterstatusupdate";
    public function __construct()
    {
        parent::__construct( TwitterStatusUpdateType::WORKFLOW_TYPE_STRING, 'Update
Twitter Status' );
    }

    public function execute( $process, $event )
    {
        $parameters = $process->attribute( 'parameter_list' );
        /* YOUR CODE GOES HERE */
        return eZWorkflowType::STATUS_ACCEPTED;
    }
}

eZWorkflowEventType::registerEventType( TwitterStatusUpdateType::WORKFLOW_TYPE_STRING,
'twitterstatusupdatetype' );

?>
```



In the above code the function you are interested in is the *public function execute(\$process, \$event)*. This function is holding your event code and is called by eZ Publish passing two parameters *\$process* and *\$event*..

Lets have a quick look at what information you can retrieve from these two parameters:

- *\$process* is an instance of the *eZWorkflowProcess* class. It is holding information relative to the eZ Publish workflow process being run such as the workflow process ID, the workflow ID, the User ID under which the workflow process is executed (see [http://pubsvn.ez.no/doxygen/4.3/html/ezworkflowprocess\\_8php\\_source.html](http://pubsvn.ez.no/doxygen/4.3/html/ezworkflowprocess_8php_source.html) for more details). This tutorial is more interested in the parameter 'parameter\_list' accessible via *\$process->attribute('parameter\_list')*. This will return the an associative array with the following keys 'object\_id', 'version', 'workflow\_id', 'trigger\_name', 'module\_name', 'module\_function', 'user\_id', 'parent\_process\_id'.
- *\$event* is an instance of the *eZWorkflowEvent* class. It is holding information relative to your workflow event. This is not useful to us at this stage.

This function should return one of several predefined values depending on the outcome of the execution of the code and what is to be done next. For example: *eZWorkflowType::STATUS\_ACCEPTED* if we should leave the workflow process and end it, *eZWorkflowType::STATUS\_DEFERRED\_TO\_CRON* if we want to defer this workflow to be processed later by a cronjob etc... See the *eZWorkflowType* class reference:

<http://pubsvn.ez.no/doxygen/4.3/html/classeZWorkflowType.html>

Our workflow event is a PHP class that extends *eZWorkflowEventType* class. Please note the class name must use the same name as our PHP file. In the constructor we are declaring our event and the name that will be displayed in drop down menus. The *execute* method is holding the actual code.

## 7 Step 4 - Develop a workflow event: Arcgo OAuth Twitter library

*Arcgo\_Service\_Twitter* aims to provide a robust but easy-to-use interface to the Twitter API. You can download the OAuth version of the library here: <http://github.com/yemkay/arcgo-twitteroauth>

What you need is the "Arcgo" and "oauth" folders located inside the downloaded archive. Copy this folder and its content into `~extension/mytwitter/lib/`

**Note: see further down this tutorial to see how to generate the key, token and secrets.**

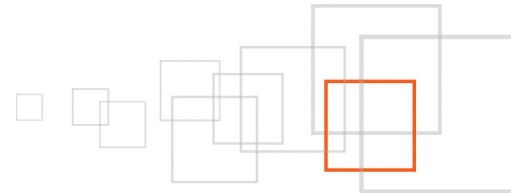
As a preliminary step we will now create an INI file to store some key variables:  
`~extension/mytwitter/settings/mytwitter.ini`

In the file we will create below, the `setup.php` script will be described futher down in this tutorial at Step 8.

```
#?ini charset="utf-8"?

[TwitterSettings]

# Optional: you can register for a Twitter API profile at: http://dev.twitter.com/
# then copy and paste the Consumer Key and Consumer Secret strings here. This will
# show your app name instead of 'eztweeter' when ever a status update is posted.
# If you choose to do so, please also update extension/mytwitter/lib/setup.php
```



```
ConsumerKey=Gq2fR3F8qPqyOariWv4csQ
ConsumerSecret=kNO49WT3xw44hovPOLQsbw2dwYHQoJ6Jr5KLyI3Y

# The following variables should be configured in your siteaccess settings
# To generate those values use the setup.php script in the lib/ folder, go to the lib/
directory of our mytwitter extension directory and run the following command in CLI
# php setup.php --register
# then copy and paste the generated URL into a browser, login into Twitter and allow
access to the eztweeter app, copy the PIN
# php setup.php --validate=[PIN]
# save the Key and Secret as the values below
#AccessToken=
#AccessSecret=

# will write debug message in var/log/common.log
#DebugOutput=enabled
```

Please note the extension of this file is ".ini". This file is a raw INI file. Its usual purpose is to declare common variables and their default values. It also allows eZ Publish to display an entry for this file when you go in the admin interface to *Setup > Ini settings > Select ini file to view*.

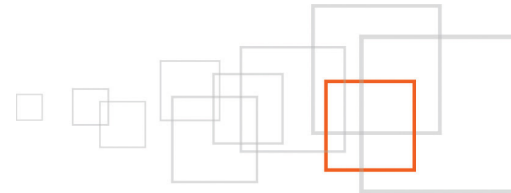
You should comment all here and create a copy of it as `mytwitter.ini.append.php` in the `siteaccess settings` folder. This follows one of the best practices in eZ Publish development and integration : never edit the default configuration files ( the ones located in `settings/* .ini` ), rather use the override system to modify a directive's value. More information on this here : <http://doc.ez.no/eZ-Publish/Technical-manual/4.4/Concepts-and-basics/Configuration>.

We will now start our code with loading some parameters from a `mytwitter.ini` file and instantiate the `Arcgo_Service_Twitter` class. For this, re-edit the file `~extension/mytwitter/eventtypes/event/twitterstatusupdate/twitterstatusupdatetype.php` and start the code after the line

```
/* YOUR CODE GOES HERE */
```

The following code is loading the INI settings and then instantiate the `Twitter ARCgo OAuth` class. Please also note the use of `eZLog` class that allows us to write debug messages to `~/var/log/common.log`

```
public function execute( $process, $event )
{
    $parameters = $process->attribute( 'parameter_list' );
    /* YOUR CODE GOES HERE */
```



```
$twitterINI = eZINI::instance( 'mytwitter.ini' );
$twitterDebugOutput = $twitterINI->variable( 'TwitterSettings', 'DebugOutput' );

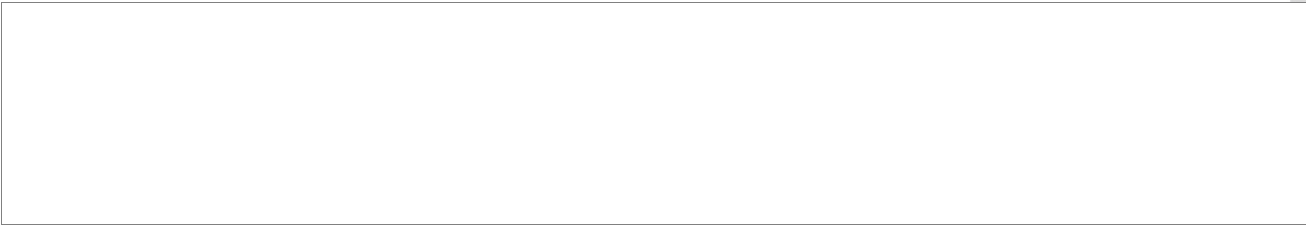
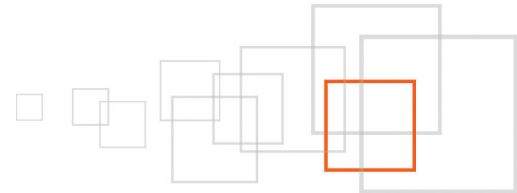
eZLog::write( "Entering eztwitter workflow" );
$twitterConsumerKey = $twitterINI->variable( 'TwitterSettings', 'ConsumerKey' );
$twitterConsumerSecret = $twitterINI->variable( 'TwitterSettings',
'ConsumerSecret' );
$twitterAccessToken = $twitterINI->variable( 'TwitterSettings', 'AccessToken' );
$twitterAccessSecret = $twitterINI->variable( 'TwitterSettings', 'AccessSecret' );

if( empty( $twitterConsumerKey ) ||
    empty( $twitterConsumerSecret ) ||
    empty( $twitterAccessToken ) ||
    empty( $twitterAccessSecret ) )
{
    if( $twitterDebugOutput == 'enabled' )
        eZLog::write( "Please configure mytwitter.ini" );
}
if( $twitterDebugOutput == 'enabled' )
    eZLog::write( "Credentials found in mytwitter.ini" );

$twitter = new Arc90_Service_Twitter();
$twitter->useOAuth( $twitterConsumerKey,
                  $twitterConsumerSecret,
                  $twitterAccessToken,
                  $twitterAccessSecret );
```

As we are using an external 3<sup>rd</sup> party library, PHP will most likely output an error message if the library is including its own PHP file even with the autoload process already updated. To fix this, we could either edit all PHP files from the library and remove all the include() or require() calls and let the autoload take care of this. Or we can also edit the config.php file at the root directory of our eZ Publish installation and set the correct 'include\_path' at the end of the file:

```
ini_set( 'include_path', ini_get('include_path'). PATH_SEPARATOR .
'/path/to/ezpublish/extension/mytwitter/lib' );
```



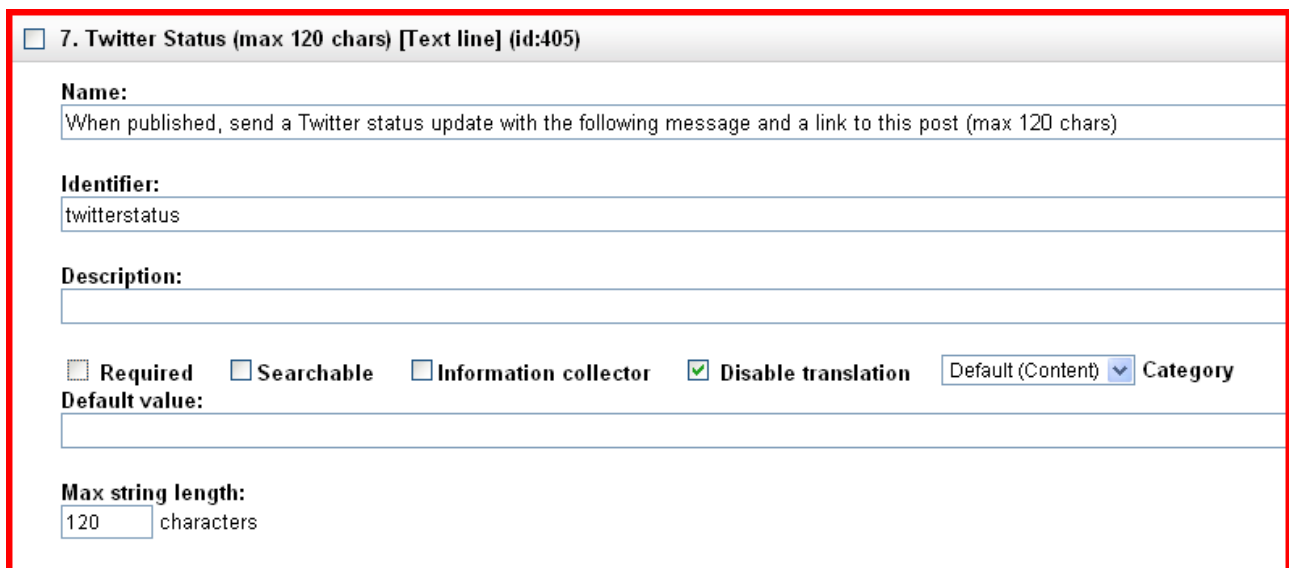
## 8 Step 5 - Develop a workflow event: content class attribute

For our workflow event to work, we need to add a new content class attribute to content classes we want to add the Twitter Status Update feature.

In Setup > Classes > Content , edit the content class “Blog post” and add a new “Text line” attribute.

Fig. 8 shows you the value of each fields. The most important ones are the “Identifier” and “Max string length”.

The “Identifier” is used by the event PHP file to extract the message, and “Max string length” is restricting the message length as it will be appended with the TinyURL and Twitter only allows 140 characters.



7. Twitter Status (max 120 chars) [Text line] (id:405)

**Name:**  
When published, send a Twitter status update with the following message and a link to this post (max 120 chars)

**Identifier:**  
twitterstatus

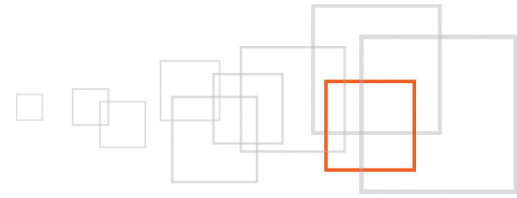
**Description:**

Required  Searchable  Information collector  Disable translation  Category

**Default value:**

**Max string length:**  
 characters

Fig. 8



## 9 Step 6 - Develop a workflow event: posting a Twitter status

The idea here is to allow the editor to add a custom message that will be tweeted. For this we will need to edit any content class we want to add the Twitter Status Update feature. Our workflow event will check for the existence of a content class attribute to pick the message from. For the moment just check the code.

First we need to retrieve the object being published. This is where the \$parameters array is useful for. Back to our twitterstatusupdate.php editing, continuation of the snippet above :

```
$objectID = $parameters['object_id'];
$object = eZContentObject::fetch( $objectID );
$nodeID = $object->attribute( 'main_node_id' );
$node = eZContentObjectTreeNode::fetch( $nodeID );
$datamap = $object->dataMap();

// Now we need to check if the content class attribute "twitterstatus"
// is present in the class and whether it is empty or not:
// Quit if attribute twitterstatus does not exist in the contentclass
if( !isset( $datamap['twitterstatus'] ) )
{
    if( $twitterDebugOutput == 'enabled' )
        eZLog::write("twitter status not found");

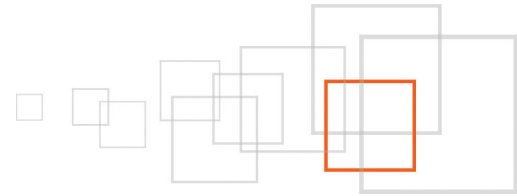
    return eZWorkflowType::STATUS_ACCEPTED;
}
if( $twitterDebugOutput == 'enabled' )
    eZLog::write( "Found twitter status attribute" );

// Else take note of it
$twitterStatus = $datamap['twitterstatus']->attribute('data_text');

if( empty( $twitterStatus ) )
{
    if( $twitterDebugOutput == 'enabled' )
        eZLog::write( "twitter status empty" );

    return eZWorkflowType::STATUS_ACCEPTED;
}
if( $twitterDebugOutput == 'enabled' )
    eZLog::write( "twitter status attribute not empty" );
```

If everything is OK, we carry on and use TinyURL API to shrink the URL to our article. This is a quick way to do it, you could also use cURL. Please note that your site.ini.append.php file needs to be configured with the



proper *SiteURL* pointing to the front end address of your website, you might need to also configure it in your admin siteaccess. This is not the best way to do this, so for a better code you might want to create a more complex code or maybe use another INI variable name so that SiteURL for your admin interface is the actual URL of the admin interface.

```
$siteINI = eZINI::instance();
$siteHost = $siteINI->variable( 'SiteSettings', 'SiteURL' );
$siteHost = preg_replace( "|/$|", "", $siteHost );

$tinyURL = eZHTTPTool::getDataByURL( 'http://tinyurl.com/api-create.php?url=' .
urlencode( "http://".$siteHost."/". $node->attribute( 'url_alias' ) ), false, false );

if( empty( $tinyURL ) )
{
    if( $twitterDebugOutput == 'enabled' )
        eZLog::write("Error with TinyURL");

    return eZWorkflowType::STATUS_ACCEPTED;
}

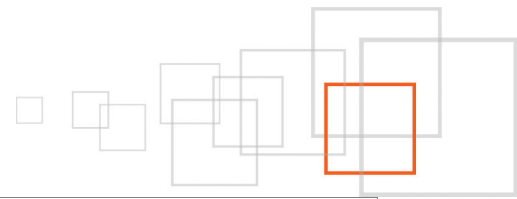
try
{
    $response = $twitter->updateStatus( "{$twitterStatus}: {$tinyURL}", 'xml' );
    if( $response->isError() )
    {
        if( $twitterDebugOutput == 'enabled' )
            eZLog::write( "Twitter error: " . $response->http_code );

        return eZWorkflowType::STATUS_ACCEPTED;
    }
    else
    {
        if( $twitterDebugOutput == 'enabled' )
            eZLog::write( "Twitter status updated: {$twitterStatus}: {$tinyURL}" );
    }
}
catch( Arc90_Service_Twitter_Exception $e )
{
    if( $twitterDebugOutput == 'enabled' )
        eZLog::write( "Error with Twitter API" );

    return eZWorkflowType::STATUS_ACCEPTED;
}

// After having sent the status update, we'll clear it so it won't be resent.
$datamap['twitterstatus']->setAttribute( 'data_text', '' );
```





```
$datamap['twitterstatus']->store();  
if( $twitterDebugOutput == 'enabled' )  
    eZLog::write( "Resetting twitter status attribute" );
```

## 10

1

## 11 Step 7 - Develop a workflow event: setup the workflow

### Activating the extension:

Now that we have our content class attribute added and have created the workflow event type script. We need to first activate our newly created extension. For this add the following to your site.ini file:

```
[EventSettings]  
ActiveExtensions[]=mytwitter
```

( more details about activating an extension here: <http://doc.ez.no/eZ-Publish/Technical-manual/4.x/Installation/Extensions/Activating-the-extension/> )

### Declaring our workflow event:

We need to tell eZ that our extension contains a workflow event by creating a workflow.ini.append.php file in the settings folder of our extension:

```
<?php /*  
[EventSettings]  
ExtensionDirectories[]=mytwitter  
AvailableEventTypes[]=event_twitterstatusupdate  
*/ ?>
```

### Regenerate the autoload array:

As we have created and used several PHP classes, you will need to regenerate the autoload array, so from command line at the eZ Publish root run:

```
php bin/php/ezpgenerateautoloads.php
```

Lets also clear the caches:

```
php bin/php/ezcache.php --clear-all -s yoursiteaccess
```

### Create the main workflow:

Create a new workflow inside the "Standard" workflow group and name it "Twitter Status Update". Now select and add "Event / Update Twitter Status" event:

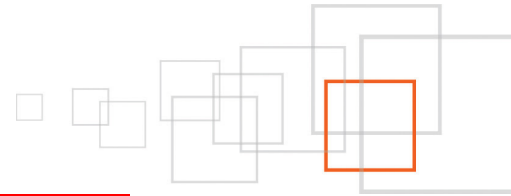


Fig. 9

**Create a post publishing workflow:**

Now, following the tutorial at Step 2, create a new workflow and name it "After publishing" workflow and add to it a new "Event / Multiplexer" event and configure it as per Fig. 10:

Fig. 10

You should have "Classes to run workflow" set to "Blog post" (you can add other classes but make sure you've added the "twitterstatus" attribute to them) and "Workflow to run" is "Twitter Status Update".

**Assigning the post publishing workflow to a trigger:**

Now click on OK and assign the "After publishing" workflow to the "content / publish / after" trigger.

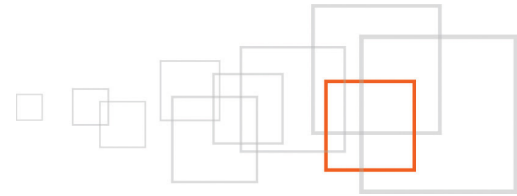
**Test:**

To test out your new workflow event, edit a blog post, fill in the Twitter Status field and click on submit.

## 12 Step 8 – Authorize eZ Publish via Twitter OAuth

Since August 2010 Twitter doesn't allow access to the service via the basic authentication method. This is affecting all scripts and bots that automatically update a Twitter account. The pain is mainly because of the way the new authentication method (OAuth) is dealing with hand shakes, it requires a human interaction in order to generate an Access Token and Access Token Secrets that will authenticate the user later one. Fortunately this is only needed once.

The Arcgo with OAuth support you have downloaded above allows a PHP script to use OAuth and update a



Twitter feed. But you need to modify few lines and create a new script allowing you to authenticate a user and generate keys and tokens for future authentication.

In Step 4 above, we have created an INI file to hold some of our variables and credentials for Twitter OAuth. It's now time to see how to retrieve them. We will need to do some small modifications to the OAuth library we downloaded earlier and create a little PHP script to help us with the Twitter authentication. This is a bit tedious because of how OAuth is working, but fortunately you only have to do it once.

#### **Modifying OAuth library to support PIN validation:**

Twitter OAuth supports two methods to validate the authorization of an app to update a user account. It can either redirect to a callback URL or generate a PIN code. For this tutorial, I've setup a Twitter customer key and customer secret string that will identify this app as 'eztweeter'. I strongly recommend you register for a Twitter API and get your own key and secret strings, this also good for your own branding as this will show your app name on the tweets. This demo Twitter API profile has been registered and configured to provide a PIN code for validation. But in order to use this PIN we need to modify the OAuth's library file 'oauth/twitterOAuth.php, edit that file and look for the function getAccessToken and modify it to look like this:

```
function getAccessToken( $token = NULL, $pin = NULL )
{
    if ( $pin )
    {
        $r = $this->oauthRequest( $this->accessTokenURL(),
                                array( "oauth_verifier" => $pin ) );
    }
    else
    {
        $r = $this->oauthRequest( $this->accessTokenURL() );
    }

    $token = $this->oauthParseResponse( $r );
    $this->token = new OAuthConsumer( $token['oauth_token'],
                                    $token['oauth_token_secret'] );

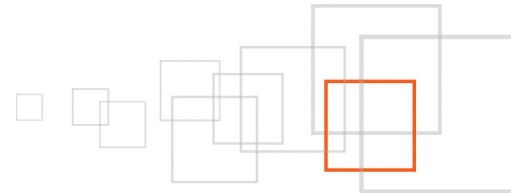
    return $token;
}
```

You now also need to create a little PHP script that will handle the authorization and validation via command line:

```
<?php
error_reporting( E_ALL | E_NOTICE );

// Go back to eZ root folder
require 'autoload.php';
//require_once( 'extension/mytwitter/lib/oauth/twitterOAuth.php' );

$cli = eZCLI::instance();
```



```
$script = eZScript::instance( array( 'description' => ( "Twitter credentials registration
/ validation\n" .
                                "Script to register and validate
OAuth credentials for Twitter\n" .
                                "\n" .
                                "extension/mytwitter/lib/setup.ph
p" ),
                                'use-session' => false,
                                'use-modules' => false,
                                'use-extensions' => true ) );

$script->startup();

// CLI parameters
$options = $script->getOptions( "[register][validate:]",
                                "",
                                array( 'register' => 'generate a
registration URL',
                                'validate' => 'validate the PIN
returned by the registration URL' ) );

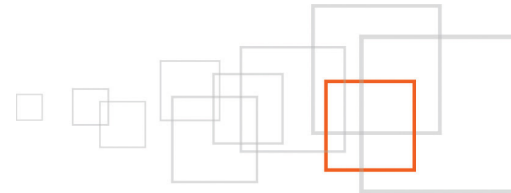
$script->initialize();

$myTwitterINI = eZINI::instance( 'mytwitter.ini' );
$consumerKey = $myTwitterINI->variable( 'TwitterSettings', 'ConsumerKey' );
$consumerSecret = $myTwitterINI->variable( 'TwitterSettings', 'ConsumerSecret' );
$accessToken = $myTwitterINI->variable( 'TwitterSettings', 'AccessToken' );

$varDir = eZSys::varDirectory();
$myTwitterTmpDir = $varDir . '/mytwitter';
if ( !file_exists( $myTwitterTmpDir ) )
{
    eZDir::mkdir( $myTwitterTmpDir, eZDir::directoryPermission(), true);
}

$noAction = true;
$register = isset( $options['register'] );
$noAction = !$register;

$pin = false;
```



```
if ( $options['validate'] )
{
    $noAction = false;
    $pin = $options['validate'];
}

if ( $register )
{
    // instantiate a TwitterOAuth object and request a token
    $oauth = new TwitterOAuth( $consumerKey, $consumerSecret );
    $request = $oauth->getRequestToken();

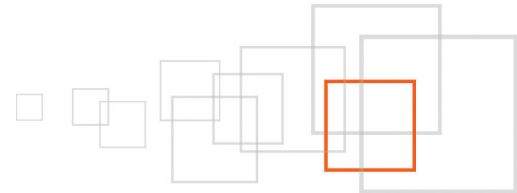
    $request_token = $request["oauth_token"];
    $request_token_secret = $request["oauth_token_secret"];

    // Saving the token in files for the validation process to use
    // Make sure to delete this after validation.
    file_put_contents( "{$myTwitterTmpDir}/request_token", $request_token );
    file_put_contents( "{$myTwitterTmpDir}/request_token_secret",
$request_token_secret );

    // Generate an authorisation URL
    $request_link = $oauth->getAuthorizeURL( $request );
    $cli->warning( "Request here: " . $request_link );
}
elseif ( $pin )
{
    // read the request token from the registration process
    $request_token = file_get_contents( $myTwitterTmpDir ."/request_token" );
    $request_token_secret =
file_get_contents( $myTwitterTmpDir ."/request_token_secret" );

    // Instantiate a TwitterOath object and provide it with the loaded token
    $oauth = new TwitterOAuth( $consumerKey, $consumerSecret,
        $request_token, $request_token_secret );

    // request an access token from Twitter
    $request = $oauth->getAccessToken( FALSE, $pin );
```



```
$cli->output( "Twitter user: {$request['screen_name']}" );

$access_token = $request['oauth_token'];
$access_token_secret = $request['oauth_token_secret'];

// Display INI file settings
$cli->output( "mytwitter.ini.append.php variables:" );
$cli->warning( "AccessToken={$access_token}" );
$cli->warning( "AccessSecret={$access_token_secret}\n" );

//require_once( "extension/mytwitter/lib/Arc90/Service/Twitter.php" );

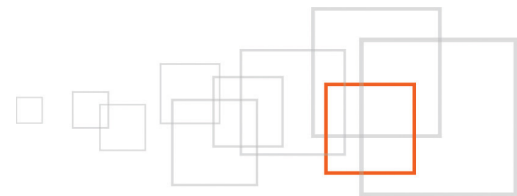
// Lets see if everything is working
$twitter = new Arc90_Service_Twitter();
// Authenticate
$twitter->useOAuth( $consumerKey, $consumerSecret, $access_token,
$access_token_secret );

// Retrieve our account's timeline
$cli->output( 'Trying to retrieve our Twitter timeline' );
$response = $twitter->getFriendsTimeline( 'json', array( 'count' => 200, 'page' =>
0 ) );
$cli->output( 'HTTP code: '.$response->getHttpCode() );

if (!$response->isError())
{
    $messages = $response->getJsonData();
    $cli->output( "Found ".count($messages). "new tweets" );
}
else
{
    $cli->output( 'Error description: '.$response->getData() );
}

// Deleting the token request files
unlink( $myTwitterTmpDir ."/request_token" );
unlink( $myTwitterTmpDir ."/request_token_secret" );

}
```



```
if ( $noAction )
{
    $cli->warning( "Please use one of the following options --register --validate. Use
--help option for more details." );
}
$script->shutdown();
?>
```

Now lets proceed with the authorization to modify your Twitter feed, change to the directory `~/extension/mytwitter/lib/` and run:

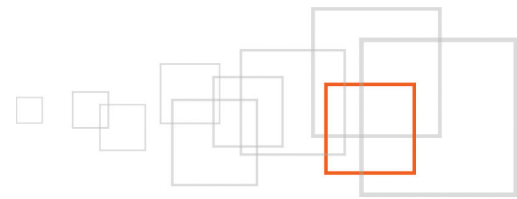
```
php setup.php --register
```

This will give you a URL that you need to open in a browser. Login into Twitter and allow eztweeter to access your feed, then copy the PIN code and run the following command:

```
php setup.php --validate=[PIN]
```

Replace [PIN] with the PIN code you copied earlier. This command will give you the INI string you need to setup in `mytwitter.ini` in the settings folder of the extension or `mytwitter.ini.append.php` of your siteaccess settings folder.

That's it.



## 13 Conclusion

Again, workflows are useful tools to setup process that occurs before or after the publication of a content. They can be as simple as the content approval process and can be much more complex and combined to the object states they can be very powerful.

We have seen in this tutorial how to setup a workflow and link it to a trigger and we applied it to a content approval process. We have then been through an example of a workflow that automatically post a status update to a Twitter account. Additionally, because of some technical restriction from Twitter, we have also developed a little CLI script which acts as a helper with Twitter's OAuth. With these examples, you should now be able to develop your own workflow events for your website needs.

## 14 Resources

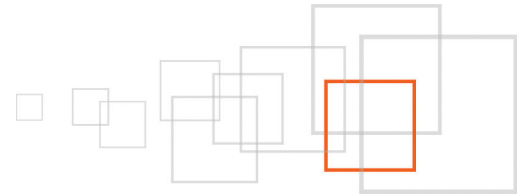
- eZ Publish workflows – Concepts and basics
- Workflow overview by Martin Bauer
- eZ Publish ezworkflowprocess class reference
- eZ Publish configuration files (.ini) – Concepts and basics
- Activating eZ Publish extensions
- eZScript class reference
- eZ Publish command line scripts
- eZSys class reference
- eZINI class reference
- eZWorkflowEventType class reference
- eZWorkflowType class reference
- eZLog class reference

## 15 Credits

Thanks to Yemkay for the Arc-twitteroauth library: <http://github.com/yemkay/arcgo-twitteroauth>

Thanks to Konstantin Kovshenin for the trick with Twitter OAuth for PHP:  
<http://kovshenin.com/archives/twitter-api-pin-based-oauth-php/>





## 16 About the author, Quoc Huy Nguyen Dinh



Huy began web development in late 1996. As he studied software engineering in France, he grew a preference for web development and gathered skills on the side with personal projects and helping a small online publishing start-up company. He discovered eZ Publish in 2008 and is now a certified eZ Publish developer working for the Financial Times in London.

## 17 License choice

GNU Free Documentation License (GFDL)

<http://www.gnu.org/copyleft/fdl.html>