Selling pay-per-download products

INTRO

Many products – PDF books, songs, and online games, among others – are well suited for a pay-per-download workflow. In such a case, customers buy products online and receive immediate access, without requiring any action from the seller. eZ Publish provides a framework to quickly set up pay-per-download products.

PAGE 1

In this article, we will set up a content structure to support products and associated downloads; install a payment gateway (eZ PayPal); write an extension to assign the appropriate permissions to buyers when they check out; and enhance the relevant templates so that buyers know where to download the files they've purchased.

Requirements

This article is intended for eZ Publish developers who are comfortable working with PHP and eZ Publish settings. The solution as a whole is not that complicated within the eZ Publish framework, but it touches upon many eZ Publish concepts. To get the most out of this article, you should first have some basic knowledge about: the content model (http://ez.no/developer/articles/an_introduction_to_ez_publish_concepts/object_oriented_content_model); sections and user permissions (http://ez.no/developer/articles/section_segmentation_and_user_permissions_part_1_of_2) (http://ez.no/developer/articles/section_segmentation_and_user_permissions_part_2_of_2); workflows (http://ez.no/doc/ez_publish/technical_manual/4_0/concepts_and_basics/workflows); and the webshop (see http://ez.no/doc/ez_publish/technical_manual/4_0/concepts_and_basics/webshop and Chapter 15 of http://ez.no/store/books/ez_publish_advanced_content_management). It would also be handy if you have experience setting up some simple extensions (http://ez.no/developer/articles/an_introduction_to_developing_ez_publish_extensions).

You should also have an eZ Publish 4.1 installation to work with, although the concepts and procedures should be similar for other versions. This article uses eZ Publish 4.1.3 with the Website Interface.

In order to properly test the installation of the eZ PayPal extension, you must have a website with a publicly accessible address. This is explained later in the article.

The buyer experience

Before we delve into any code or even content, let's consider how we want the process to look from the buyer perspective.

1. Buyer browses products and adds some to his/her online shopping cart
2. Buyer initiates checkout process and is prompted to pay for the product
3. Upon successful payment, buyer is presented with relevant download links on the confirmation page and in the confirmation e-mail.

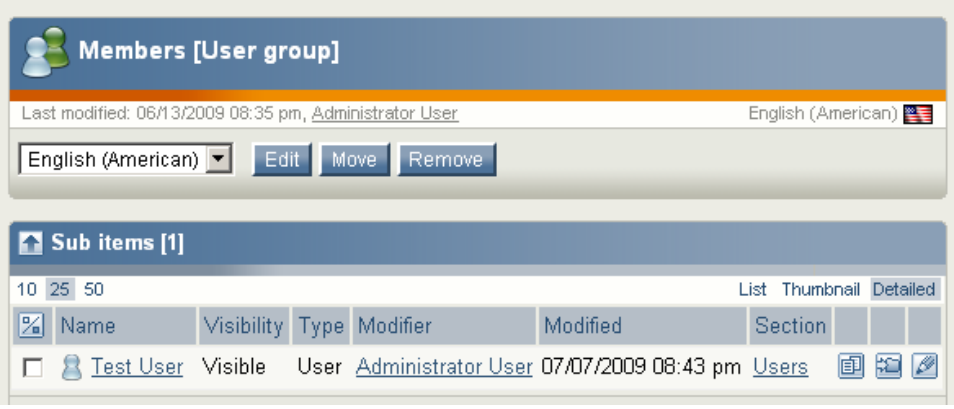
This is rather straightforward from the buyer's perspective. Let's now consider what we will need to do to achieve this.

- Associate the actual downloads with products: We will do this by adding a related object attribute to one of our product classes.
- Shopping cart and checkout process: This is a built-in eZ Publish feature and we will use most of the defaults here.
- Payment process: We will install an existing PayPal payment gateway extension, making use of the eZ Publish workflow system.
- Download permissions: We will write an extension to assign permissions at checkout so that the buyer can access the files associated with the purchased products
- Confirmation page and e-mail: We will modify existing eZ Publish templates to add the necessary download information.

Setting up the product and permissions structure

Our example site will sell PDF books, giving buyers the advantage of getting the book cheaper and faster, and removing the need for us to ship the books.

In preparation for the following steps and to ease the testing process, log in to the Administration Interface as an Administrator user, and create a user under the Members user group. This article will name this user "Test User". This user will be your test buyer.

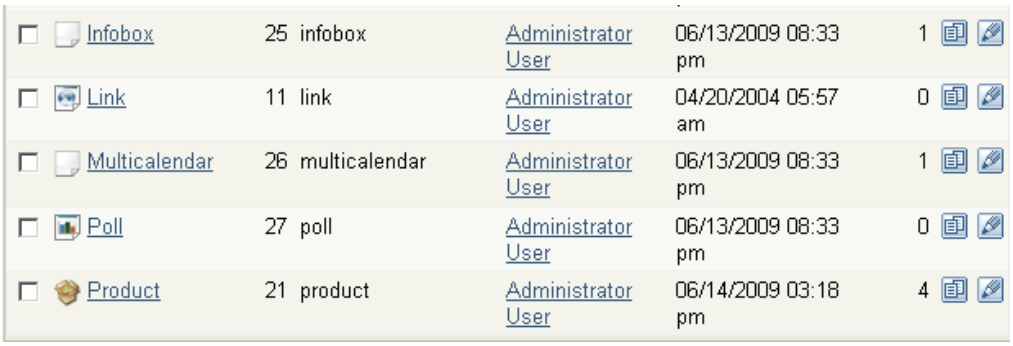


Product class

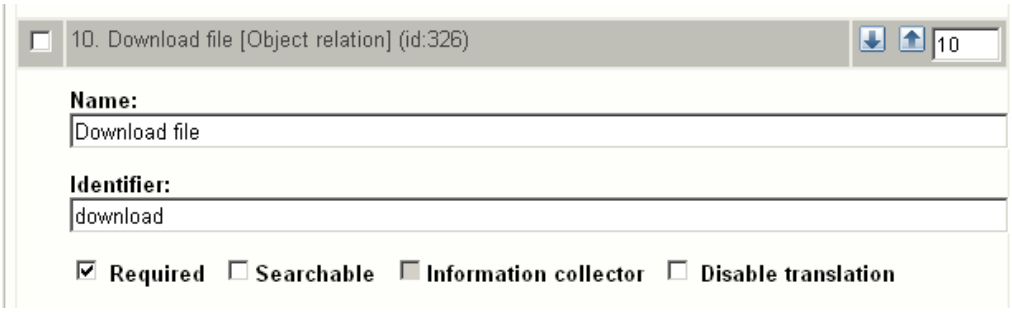
A default Website Interface installation comes with a basic "Product" class, consisting of at least a name, a description, an image, and a price. The "Price" attribute is what makes the content object a recognized product in the webshop, and the other attributes are useful for descriptive purposes.

For simplicity, we will assume that every product on our site represents a digital download. Therefore, we will modify the existing "Product" class in order to associate a file with each Product object.

Sign in to the Administration Interface using the Administrator user account and go to the "Setup > Classes" page. Then, click to view the classes in the "Content" class group. In the list of classes, click the "Edit" icon for the "Product" class.



At the bottom of the "Class edit" interface for the "Product" class, add a new attribute of the "Object relation" type. Name the attribute "Download file" with an identifier "download". For our purposes, we will make this attribute required but not searchable.

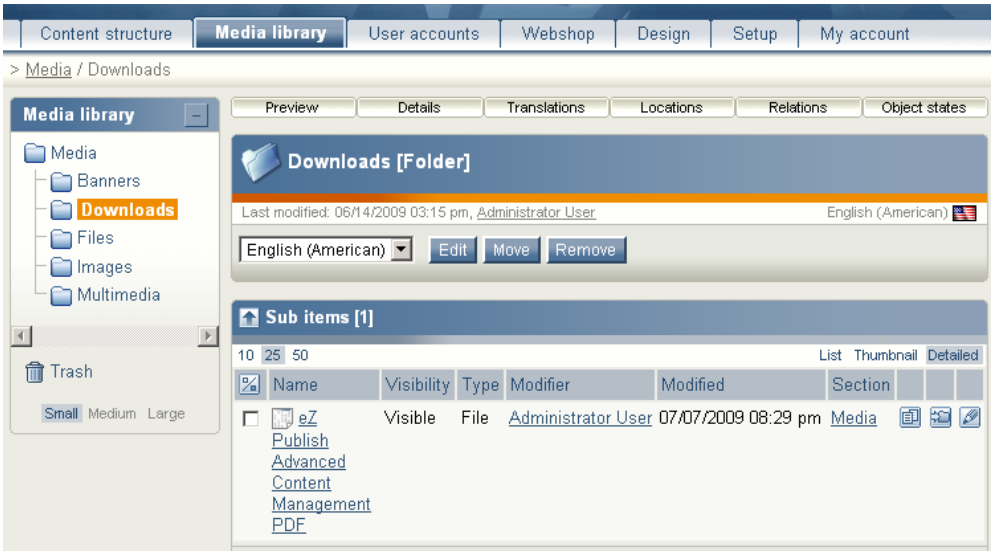


The reason why we are using an attribute of the "Object relation" datatype is so that we can store the actual files in a location of the content tree separate from products. The products can be viewed by everybody, but access to the download files will, by default, be prohibited to the general public and granted to buyers as needed.

Location for download files

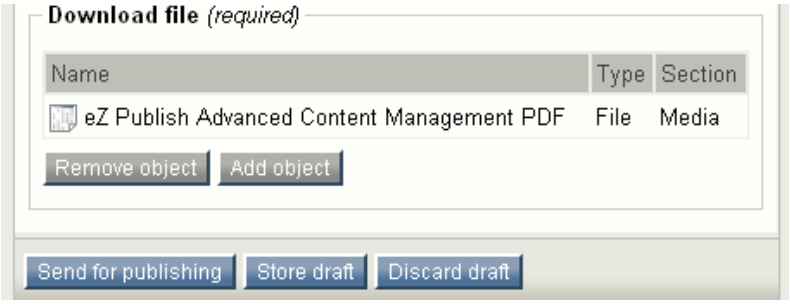
A good place to put the download files is in the media library, as the general public cannot browse that section through the front-end of the site, and the fact that it is separate from the rest of the content makes for better organization. Therefore, create a "Folder" object called "Downloads" in the media library.

Under the "Downloads" folder, create a "File" object, uploading the file for a book you are selling.

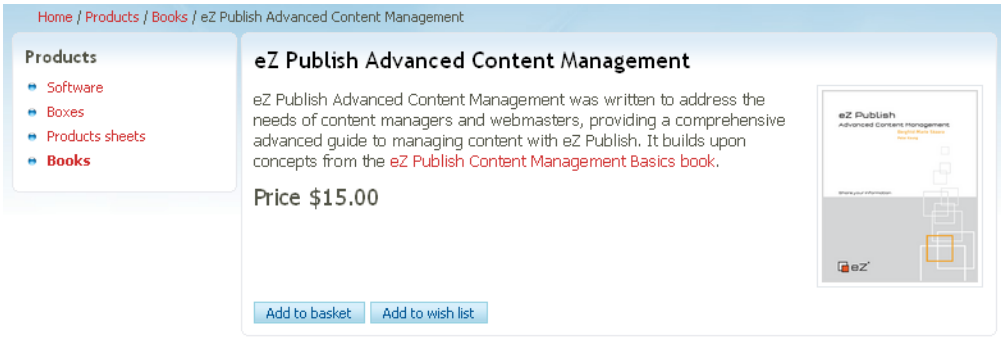


Create a product object

Since we are using a Website Interface install we can use the existing products location and templates. Browse to the "Content structure" tab in the Administration Interface, click to the "Products" folder, then create a sub-item folder called "Books". Under the "Books" folder, create a "Product" object. Give it whatever name, description, price, and image you want. When it comes to the "Download file" attribute, add the "File" object you created above as the related object.



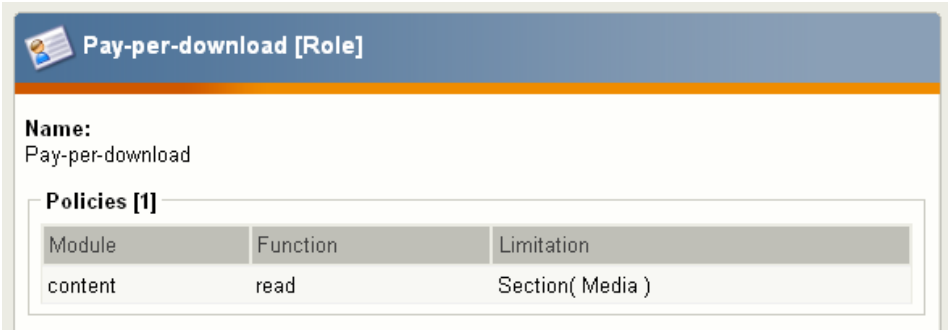
Once you have published your book product, view it on the front-end of the site, noting that the download file is not used by the template and thus not publicly displayed.



Now, log in to the front-end of the site using the Test User account you previously created. If you try to access the file directly on the front end (through a URL such as "yoursite.com/Media/Downloads/eZ-Publish-Advanced-Content-Management-PDF"), you should get an "Access denied" error.

Pay-per-download role

Now, let's create create a role that will provide access to the files associated with purchased products. Under the "User accounts" tab in the Administration Interface, click the "Roles and policies" link in the left menu. Then, create a new role called "Pay-per-download" that has a policy giving permission to the "read" function of the "content" module under the "Media" section.



This is a broad role that we will assign per-user, per-product using the subtree limitation. This assignment will be done automatically at the end of the checkout process. However, you can test this by assigning the role to the "Test User" account, with a subtree limitation of the "File" object you created.

Then, see whether that Test User user can access "yoursite.com/Media/Downloads/eZ-Publish-Advanced-Content-Management-PDF". When you have confirmed this, remove the role assignment.

Now that we've set up the content and permission structure for our pay-per-download functionality, we can move on to the next step, which is installing a payment gateway.

Payment process

As there is already an in-depth article on installing a payment gateway (specifically, for eZ Authorize: http://ez.no/developer/articles/installing_an_ez_publish_payment_gateway), this article will skip most of the theory and focus on the practical steps.

In our example, we are requiring buyers to pay via PayPal before they can access the downloadable products. The eZ PayPal account is a workflow event that will be triggered at the correct step in the checkout process. It will handle the payment process on the PayPal site and return to the example site upon successful payment.

PayPal Sandbox account

When setting up any sort of PayPal integration with a website, it is useful to sign up for a free PayPal Sandbox (https://developer.paypal.com/) account. With such an account, you can create a test buyer and test seller account and buy all the example products you want! Make sure you have a PayPal Sandbox account before continuing, as you will need to use some test PayPal accounts shortly.

Note that in order to properly test the PayPal integration, you must also do your testing on a website that has a publicly available address (that is, it cannot just be "localhost") in order for the PayPal website to be able to communicate back.

Installing and configuring the eZ PayPal extension

The eZ PayPal extension can be downloaded at http://svn.ez.no/svn/extensions/ezpaypal/trunk/

Place the contents of that "trunk" folder into the folder extension/ezpaypal in your eZ Publish installation. Then, activate the extension globally in settings/override/site.ini.append.php, adding it to the "ActiveExtensions" list:

```
[ExtensionSettings]
ActiveExtensions[]
ActiveExtensions[]=ezpaypal
```

The eZ PayPal extension also has some of its own settings in extension/ezpaypal/settings/paypal.ini. The only ones that we need to worry about for now are the PayPal server name (which we will set to the PayPal Sandbox server) and the test PayPal seller account:

```
[ServerSettings]
#ServerName=https://www.paypal.com
ServerName=https://www.sandbox.paypal.com
RequestURI=/cgi-bin/webscr

# field: "business"
# e-mail of receiver
Business=yoursandboxaccount@blah.com
```

A final requirement of the eZ PayPal extension is that you force buyers on your site to have eZ Publish

user accounts. To do so, make sure that the webshop account handler is set to "ezdefault" in settings/override/shopaccount.ini.append.php:

```
[AccountSettings]
Handler=ezdefault
```

To ensure that all of these configurations take effect, clear all caches, either from the right side of the Administration Interface, or on the command line from the root of your install:

```
php bin/php/ezcache.php –clear-all
```

As is the usual process with eZ Publish 4.0 and higher, you must now re-generate the autoloads file, either on the Setup > Extensions page in the Administration Interface, or on the command line from the root of your install:

```
php bin/php/ezpgenerateautoloads.php -e -p
```

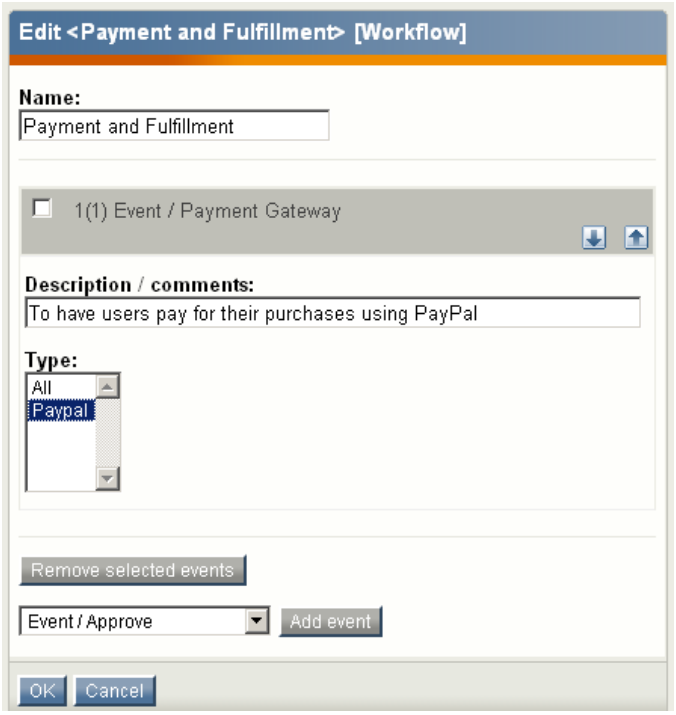
("-e" to generate the extension autoloads and "-p" to show a nice progress bar)

Setting up the eZ PayPal workflow

Although the eZ PayPal extension is now activated and ready to use, you must also instruct eZ Publish on when to actually run it. This is where we will plug it into the workflow system. This involves creating a workflow group, creating the workflow, and assigning it to run at the appropriate trigger; all of this is done in the Administration Interface.

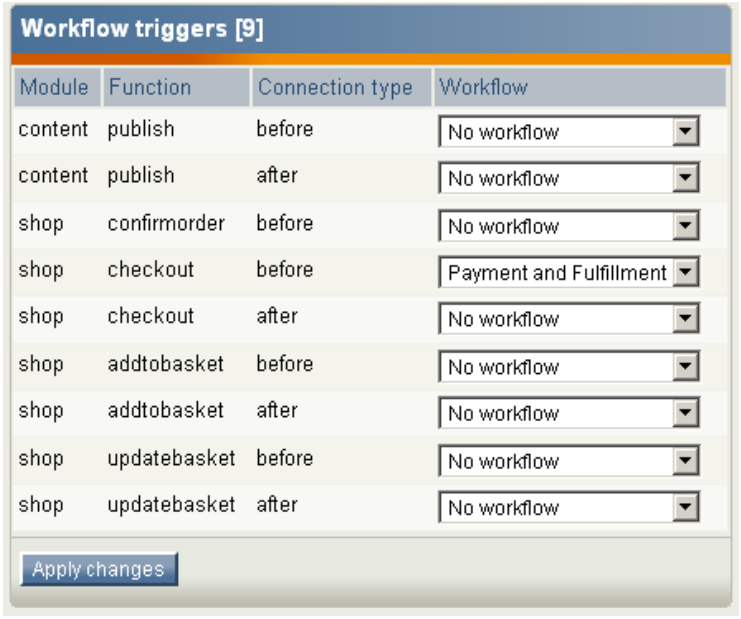
First, on the "Setup > Workflows" page, click the "New workflow group" button. Name the workflow group "Payment gateways". Workflow groups serve only to organize workflows.

When viewing the new workflow group, click the "New workflow" button. Name the workflow "Payment and Fulfillment", select "Event / Payment Gateway" from the dropdown list, and click the "Add event" button.



Select the "Paypal" type, add a useful description, then click the "OK" button.

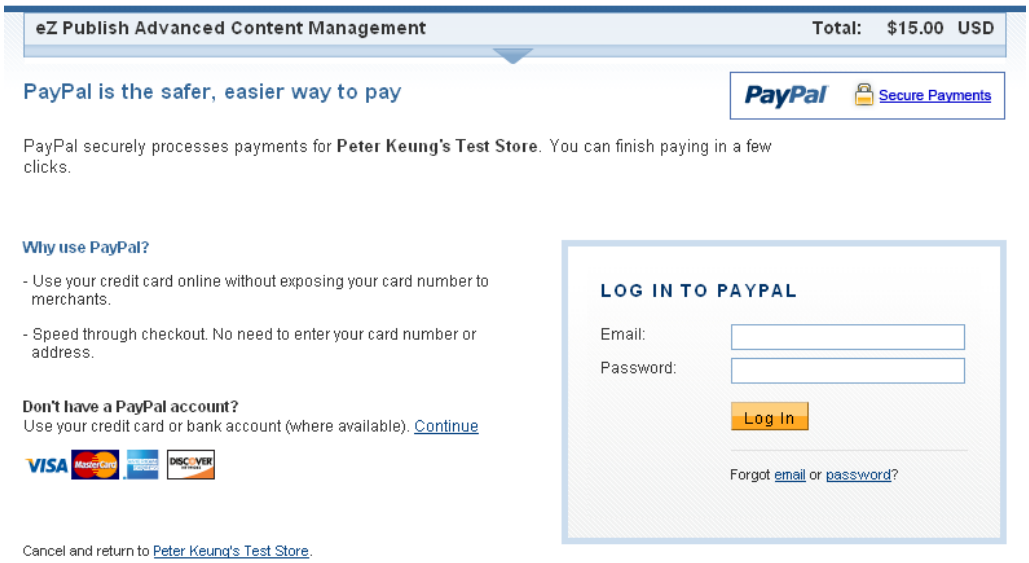
Your new workflow can now be assigned to the appropriate trigger. Click the "Triggers" link in the left menu, then select "Payment and Fulfillment" as the workflow to run before the "checkout" function of the "shop" module.



More testing

To make sure that your PayPal workflow is running properly, you can buy your PDF product using your Test User account. However, first log in to your PayPal Sandbox account.

After you add the PDF product to the shopping cart, proceed to check out, and click the "Confirm" button, you should be redirected to the PayPal Sandbox site, prompting you to log in and pay for the product (using your test PayPal account).



Of course, after you complete the payment and transaction, your "Test User" eZ Publish user should still not have access to download the associated file – something we will work on next.

Extension: Assigning permissions upon purchase part 1

With the permissions structure in place, we need to create the workflow event that will assign buyers the correct role with the correct limitation when they have checked out. We need to create a workflow extension that will be triggered after the buyer has paid.

The framework for a workflow extension is quite straightforward, including the following extension file and folder structure (beneath eZ Publish's "extension" folder):

```
payperdownload
-- eventtypes
---- event
------ payperdownload
-------- payperdownloadtype.php
-- settings
---- workflow.ini.append.php
---- payperdownload.ini
```

workflow.ini

In workflow.ini.append.php within your extension, you need to refer to the extension directory itself as containing (a) workflow event(s), and specify the name of the event, prefixed by "event_":

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

[EventSettings]
ExtensionDirectories[]=payperdownload
AvailableEventTypes[]=event_payperdownload

*/ ?>
```

payperdownloadtype.php

payperdownloadtype.php will contain all of the code you want to execute when the workflow is triggered. For now, we will just set up the framework for the class using some boilerplate code:

```
<?php

class PayPerDownloadType extends eZWorkflowEventType
{
    const WORKFLOW_TYPE_STRING = "payperdownload";

    public function __construct()
    {
        parent::__construct( self::WORKFLOW_TYPE_STRING,
ezi18n( 'kernel/workflow/event', 'Pay Per Download' ) );
        $this->setTriggerTypes( array( 'shop' => array( 'checkout' =>
array( 'before' ) ) ) );
    }

    public function execute( $process, $event )
    {
        // Important code will go here

        return eZWorkflowType::STATUS_ACCEPTED;
    }
}
```

```
eZWorkflowEventType::registerEventType( PayPerDownloadType::WORKFLOW_TYPE_STRING,
'PayPerDownloadType' );
?>
```

The "PayPerDownloadType" function defines the workflow event and specifies the trigger(s) for which it is meant. The "execute" function will contain all of our important code to assign the role based on the product(s) purchased. We will write this code shortly. The last line in the file registers our event with eZ Publish.

For now our code does nothing, of course, but we should first make sure that our framework works before we proceed.

Activate the extension

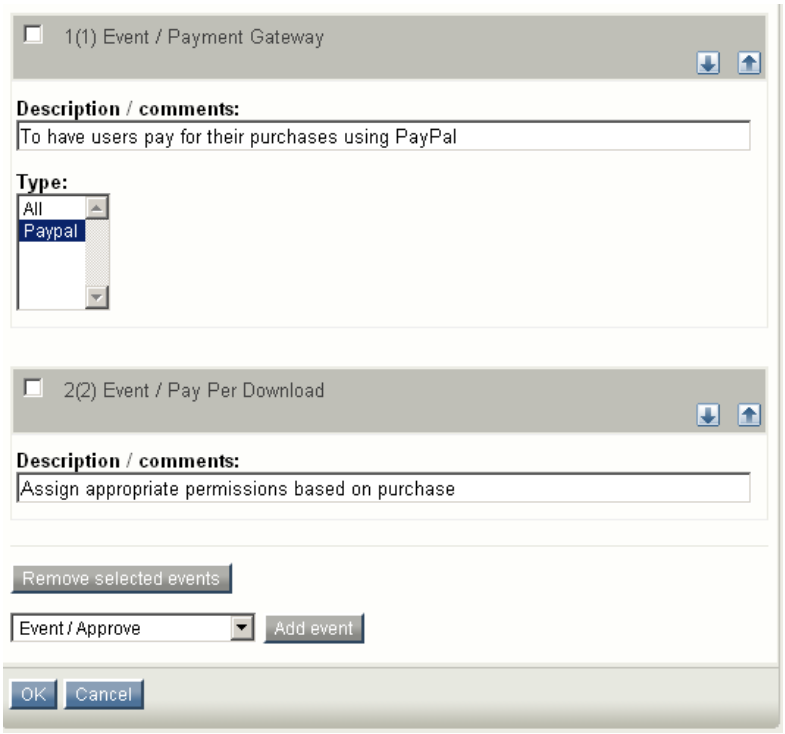
The extension is activated as is the normal process in site.ini.append.php, and we will do this in that file in settings/override:

```
[ExtensionSettings]
ActiveExtensions[]
# ... other extensions are also activated in this list
ActiveExtensions[]=payperdownload
```

You should also clear the cache and regenerate the autoloads file as was described for the eZ PayPal extension.

Tell eZ Publish when to use the workflow

We want our event to directly follow the PayPal payment event, and will actually use the same workflow. Go to "Setup > Workflows" in the Administration Interface click the "Payment gateways" workflow group, then edit the "Payment and Fulfillment" workflow. Add an event of the "Event / Pay per download" type in this workflow.



As our workflow event doesn't really do anything yet, you shouldn't notice any difference when you use your "Test User" account to purchase a product. However, if you were able to add an event of the "Pay Per Download" type in the Administration Interface, that means that eZ Publish has recognized your workflow extension.

Extension: Assigning permissions upon purchase part 2

Workflow code in the "execute" function

Now we will program the heart of our workflow extension, using the eZ Publish API <http://pubsvn.ez.no/doxygen/> and the variables that were passed to the "execute" function (in "eventtypes/events/payperdownload/payperdownloadtype.php" in your extension) to perform the role assignment(s). The process of writing this code from scratch would involve browsing through the API and piece by piece code testing (using functions like var_dump(), die(), eZExecution::cleanExit() <http://pubsvn.ez.no/doxygen/4.1/html/classeZExecution.html#58d7214394cc60d1bc84e781903e7ae> as needed), which we will not delve into. However, if you are a relative beginner to the eZ Publish API, it is worth browsing through the class lists and definitions to see all that is possible, as you have even more possibilities when you're calling eZ Publish classes in PHP land than with eZ Publish template operators. The classes are named intuitively, and the methods are grouped well and explained in quite a lot of detail. Be sure to take the time to explore the classes, methods, and attributes used in this article.

Role assignment(s)

The last part of our code is where we will start. Before the workflow is accepted and finished running, we do the actual role assignment(s) using the eZRole::assignToUser <http://pubsvn.ez.no/doxygen/4.1/html/classeZRole.html#a07120c612a36a400e5a3758390606c0> method:

```
$role = eZRole::fetch( $roleID );
$role->assignToUser( $userID, $limitIdentifier, $limitValue);

eZRole::expireCache();
```

The first line creates an eZRole object <http://pubsvn.ez.no/doxygen/4.1/html/classeZRole.html#2cee5a342b5a32de49a874fb738aaef2> when supplied a role ID. The second line assigns that role to a specified user, with a given limitation type ("Subtree" or "Section"; this will be "Subtree" in our case), and the actual limitation value, which is a path of node IDs to the download file. The last line clears the role cache.

There is quite a bit more code to go, but it is all related to supplying the correct values for the variables used in the parameters above:

- role ID
- user ID
- limitation type
- limitation value (the limitation value being the most involved to find)

INI and other basic settings

There are a few variables that will be consistent for every pay-per-download role assignment. We can define these in an INI file and then call those values in our workflow event. In "settings/payperdownload.ini" within your extension, enter the following:

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

```
[PayPerDownloadSettings]
# Which role contains the policy for pay-per-download?
RoleID=6
# Which content classes (by identifier) contain pay-per-download files?
ContentClasses[]
ContentClasses[]=product
# Which attribute (by identifier) of the above class contains the pay-per-download
file?
Attributes[]
Attributes[product]=download

*/ ?>
```

If you used the same basic Website Interface install that this article used, your "Pay per download" role with have an ID of 6. Be sure to double check what the relevant role ID is on your installation by viewing the role in the Administration Interface. The URL should end in "role/view/<role_id>".

We will access these settings, as well as define the limitation type and user ID, near the top of our "execute" function. Our function should now look like this:

```
    public function execute( $process, $event )
    {
        $ini = eZINI::instance( 'payperdownload.ini' );

        // Which role contains the policy for pay-per-download?
        $roleID = $ini->variable( 'PayPerDownloadSettings', 'RoleID' );

        // Which content class (by identifier) contains the pay-per-download file?
        $contentClasses = $ini->variable( 'PayPerDownloadSettings',
'ContentClasses' );

        // Which attribute (by identifier) of the above class contains the pay-per-
download file?
        // Note that this is in an array because later we will pass it to
fetchAttributesByIdentifier
        $attributes = $ini->variable( 'PayPerDownloadSettings', 'Attributes' );

        // We want the limit identifier to be by subtree
        $limitIdentifier = 'Subtree';

        // Get the current user
        $userID = $process->UserID;

        /* Not going to use these yet

        // Create the role object
        $role = eZRole::fetch( $roleID );

        // Assign the role
        $role->assignToUser( $userID, $limitIdentifier, $limitValue);

        // Clear the role cache
        eZRole::expireCache();

        */

        return eZWorkflowType::STATUS_ACCEPTED;
```

```
    }
```

Note the appropriate calls to create an instance of our INI file <http://pubsvn.ez.no/doxygen/4.1/html/classeZINI.html#7ad0c6a1340f500aa97f2c24677f6942> and to access the variables within the INI file <http://pubsvn.ez.no/doxygen/4.1/html/classeZINI.html#b5e9a1c407549d857f36d55c1c29c131>. We were also able to access the user ID from the $process object, which is supplied as part of the standard workflow event framework.

From the list of variables that we need to do the role assignment, we are left with the undefined limitation value.

Getting the limitation value

Recall the the limitation value will be a path of node IDs to the download file, so that the user can download the file associated with each product. Let's take a moment to map out how we are going to get that path:

- Access the current order information
- Find out what products were purchased
- Find out which (if any) products are of the classes for downloadable products
- Loop through the relevant products and access the file associated with each product
- Find out the node ID path for each file

Information about the current order is also available in the already supplied $process object, as you will see below. The rest of the code is explained within the comments.

```
[...]

        // Get the current user
        $userID = $process->attribute( 'user_id' );

        // Get the order ID so that we can find out what objects there were
        $parameters = $process->attribute( 'parameter_list' );
        $orderID = $parameters['order_id'];

        // Get the order
        $thisOrder = eZOrder::fetch( $orderID );

        // Create the role object
        $role = eZRole::fetch( $roleID );

        // Loop through each product to see whether it's relevant for role
assignment
        foreach ($thisOrder->productItems() as $thisProduct)
        {
            $classIdentifier = $thisProduct["item_object"]->ContentObject-
>attribute( 'class_identifier' );

            // Is this in the list of downloadable products?
            if( in_array( $classIdentifier, $contentClasses ) )
            {
```

```
                // We have a match, so the last thing we need to do is to fetch the
node of the file
                // First we want to grab the object so that we can get at its
attributes
                $thisObject = $thisProduct["item_object"]->ContentObject;
                $dataMap = $thisObject-
>fetchAttributesByIdentifier( array( $attributes[$classIdentifier] ) );

                // There should only be one $dataMap item, so get the path of that
                foreach( $dataMap as $dataMapAttribute)
                {
                    $node =
eZContentObjectTreeNode::fetchByContentObjectID($dataMapAttribute-
>attribute( 'data_int' );
                    // We're only after the main node
                    $limitValue = $node[0]->attribute( 'path_string' );
                }

                // Assign the role
                $role->assignToUser( $userID, $limitIdentifier, $limitValue);
            }
        }

        // Clear the role cache
        eZRole::expireCache();

[...]
```

The additional API code used above involves the following classes:

- eZOrder <http://pubsvn.ez.no/doxygen/4.1/html/classeZOrder.html>: to fetch the order, and fetch the product items, which gave us eZContentObject objects

- eZContentObject <http://pubsvn.ez.no/doxygen/4.1/html/classeZContentObject.html>: to find out whether the class identifier for each product matched the list of defined downloadable products (although in our example every product would have a downloadable file); and to access the "Download file" attribute to get at the relevant file node(s)

- eZContentObjectTreeNode <http://pubsvn.ez.no/doxygen/4.1/html/classeZContentObjectTreeNode.html>: to access the path to the relevant file node(s)

Notice that the limitation value for each downloadable file is finally defined here:

```
// We're only after the main node
$limitValue = $node[0]->attribute( 'path_string' );
```

The role assignment is then performed for each downloadable file.

Testing the complete workflow event code

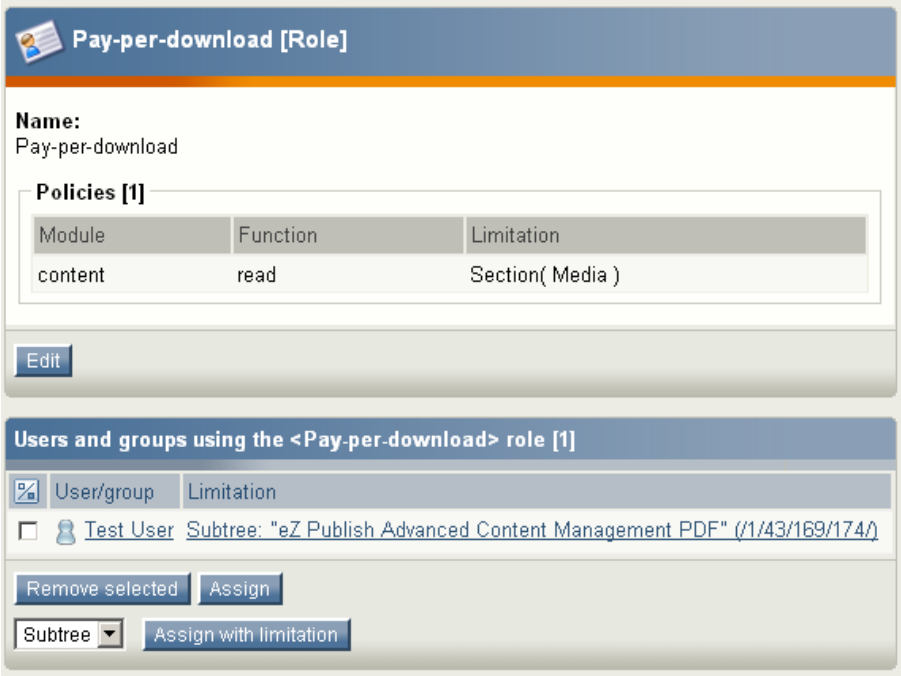
Your complete "execute" function should now look like this:

```
    public function execute( $process, $event )
    {
```

```
        $ini = eZINI::instance( 'payperdownload.ini' );

        // Which role contains the policy for pay-per-download?
        $roleID = $ini->variable( 'PayPerDownloadSettings', 'RoleID' );

        // Which content class (by identifier) contains the pay-per-download file?
        $contentClasses = $ini->variable( 'PayPerDownloadSettings',
'ContentClasses' );

        // Which attribute (by identifier) of the above class contains the pay-per-
download file?
        // Note that this is in an array because later we will pass it to
fetchAttributesByIdentifier
        $attributes = $ini->variable( 'PayPerDownloadSettings', 'Attributes' );

        // We want the limit identifier to be by subtree
        $limitIdentifier = 'Subtree';

        // Get the current user
        $userID = $process->attribute( 'user_id' );

        // Get the order ID so that we can find out what objects there were
        $parameters = $process->attribute( 'parameter_list' );
        $orderID = $parameters['order_id'];

        // Get the order
        $thisOrder = eZOrder::fetch($orderID);

        // Create the role object
        $role = eZRole::fetch( $roleID );

        // Loop through each product to see whether it's relevant for role
assignment
        foreach ($thisOrder->productItems() as $thisProduct)
        {
            $classIdentifier = $thisProduct["item_object"]->ContentObject-
>attribute( 'class_identifier' );

            // Is this in the list of downloadable products?
            if( in_array( $classIdentifier, $contentClasses ) )
            {

                // We have a match, so the last thing we need to do is to fetch the
node of the file
                // First we want to grab the object so that we can get at its
attributes
                $thisObject = $thisProduct["item_object"]->ContentObject;
                $dataMap = $thisObject-
>fetchAttributesByIdentifier( array( $attributes[$classIdentifier] ) );

                // There should only be one $dataMap item, so get the path of that
                foreach( $dataMap as $dataMapAttribute)
                {
                    $node =
eZContentObjectTreeNode::fetchByContentObjectID( $dataMapAttribute-
>attribute( 'data_int' ) );
                    // We're only after the main node
                    $limitValue = $node[0]->attribute( 'path_string' );
                }
```

```
                // Assign the role
                $role->assignToUser( $userID, $limitIdentifier, $limitValue);
            }
        }

        // Clear the role cache
        eZRole::expireCache();

        return eZWorkflowType::STATUS_ACCEPTED;
    }
```

You can now test your workflow event by buying the book product you created. Use the "Test User" account on the front-end, add the product to your shopping cart, initiate the checkout process, then pay for the product using your PayPal Sandbox test buyer account. Then, log in to the Administration Interface using your Administrator user account, navigate to the "Pay-per-download" role, and see that it has been assigned correctly. The "Test User" should be using the "Pay-per-download" role, limited to the download file.



You can also test this by seeing if the Test User account can access "yoursite.com/Media/Downloads/eZ-Publish-Advanced-Content-Management-PDF".

Confirmation e-mails and pages

We tested that our role assignment works by having our "Test User" account purchase a product and manually access the path to the download file node from the media library. However, this is not a reasonable use case. At the very least, we'd want to give the buyer links to download their purchases. And to be more user friendly, the links should lead directly to the downloadable file – not to an intermediary page.

Note: In this section, we will discuss modifying certain templates. However, when modifying any existing templates (from, for example, the Website Interface design or the standard design), it is best practice to create your own design extension containing copies that take precedence over the original templates. This is a straightforward process and you should see the How to Create eZ Publish Extensions article for an outline <http://ez.no/developer/articles/an_introduction_to_developing_ez_publish_extensions>.

Confirmation e-mail template

When a buyer has completed the checkout process, eZ Publish will e-mail the order details to the buyer and to the site administrator. Within this e-mail is an itemized list of purchases:

```
Product items
1x eZ Publish Advanced Content Management $15.00: $15.00

Subtotal of items: $15.00
```

Beneath each downloadable item, we will add a download link. The order e-mail template in question is located at design/base/templates/shop/orderemail.tpl. We will copy this file to our design extension in order to edit it. Currently, the code to loop through each purchase is as follows:

```
{section name=ProductItem loop=$order.product_items show=$order.product_items
sequence=array(bglight,bgdark)}
{$ProductItem:item.item_count}x {$ProductItem:item.object_name}
{$ProductItem:item.price_inc_vat|l10n( 'currency', $locale, $symbol )}:
{$ProductItem:item.total_price_inc_vat|l10n( 'currency', $locale, $symbol )}

{/section}
```

We will replace this code with this:

```
{foreach $order.product_items as $item}
{$item.item_count}x {$item.object_name} {$item.price_inc_vat|l10n( 'currency',
$locale, $symbol )}: {$item.total_price_inc_vat|l10n( 'currency', $locale,
$symbol )}
{def $download_classes = ezini('PayPerDownloadSettings', 'ContentClasses',
'payperdownload.ini')}
{if $download_classes|contains($item.item_object.contentobject.class_identifier)}
    {def $download_attributes = ezini('PayPerDownloadSettings', 'Attributes',
'payperdownload.ini')}
    {def $download_attribute =
$download_attributes[$item.item_object.contentobject.class_identifier]}

    - Download: {concat('http://', ezini('SiteSettings', 'SiteURL', 'site.ini'), '/
content/download/',$item.item_object.contentobject.data_map.
$download_attribute.content.id,'/',$item.item_object.contentobject.data_map.
$download_attribute.content.data_map.file.id)|ezurl('no')}
    {undef $download_attributes}
```

```
    {undef $download_attribute}
{/if}

{/foreach}
```

The code uses very similar logic to the workflow extension that we created in order to loop through each purchased product, see whether it is a downloadable product, and if so, access the node of the download file. Here, we are building a link to the "download" view of the "content" module, which is a special view that prompts the user through their browser to either download or save the file instead of showing a page. That view takes two parameters: the ID of the object in question, and the ID of the attribute containing the file. This is more efficient, user-friendly, and secure than linking to the page representing the download file's node in the media library. The e-mail text should now look something like this:

```
Product items
1x eZ Publish Advanced Content Management $15.00: $15.00
    - Download: http://www.yoursite.com/training/content/download/172/1028

Subtotal of items: $15.00
```

Order confirmation page

The same download link from the confirmation e-mail should also be given on the order confirmation page (the last page shown in the checkout process). The default file used, and to be overridden in a design extension, is located at "extension/ezwebin/design/ezwebin/templates/shop/orderview.tpl". The existing code loops through each purchased product in a table:

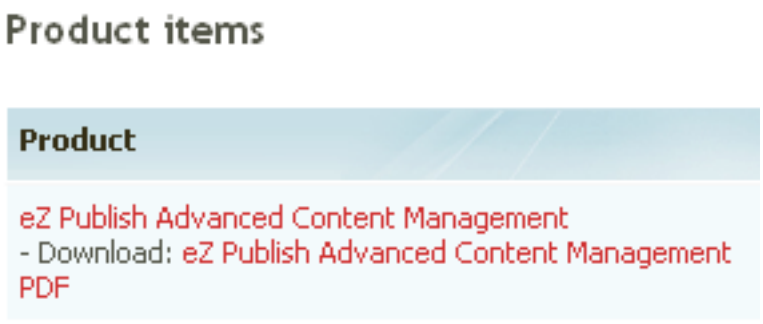
```
{foreach $order.product_items as $product_item sequence array( 'bglight',
'bgdark' ) as $style}
<tr class="{$style}">
    <td>
    <a href={concat( "/content/view/full/", $product_item.node_id )|
ezurl}>{$product_item.object_name}</a>
    </td>
```

We will add to that code to show the "content/download" links for each downloadable product:

```
{foreach $order.product_items as $product_item sequence array( 'bglight',
'bgdark' ) as $style}
<tr class="{$style}">
    <td>
    <a href={concat( "/content/view/full/", $product_item.node_id )|
ezurl}>{$product_item.object_name}</a>
    {def $download_classes = ezini('PayPerDownloadSettings', 'ContentClasses',
'payperdownload.ini')}
    {if $download_classes|
contains($product_item.item_object.contentobject.class_identifier)}
        {def $download_attributes = ezini('PayPerDownloadSettings', 'Attributes',
'payperdownload.ini')}
        {def $download_attribute =
$download_attributes[$product_item.item_object.contentobject.class_identifier]}
        <br />
        - Download: <a href={concat('content/download/',
$product_item.item_object.contentobject.data_map.
$download_attribute.content.id,'/',
$product_item.item_object.contentobject.data_map.
```

```
$download_attribute.content.data_map.file.id)|
ezurl()}>{$product_item.item_object.contentobject.data_map.
$download_attribute.content.name}</a>
        {undef $download_attributes}
        {undef $download_attribute}
    {/if}
    </td>
```

This will product a result similar to the screenshot below:



The pay-per-download solution is now complete!

Extra considerations

This article has provided a working framework for a pay-per-download solution on eZ Publish. However, there are many ways that you could improve or customize it to suit your needs.

Order history

The eZ Publish "shop" module has a view for an order history intended for site administrators to view all orders made on the site. Therefore, it is not suitable to allow normal users to access this view. However, you can create your own module and view based on what eZ Publish already provides, in order to show a user only their order history. A sample module called "orderhistory" is provided in the sample extension download.

Order list

ID	Date	Total ex. VAT	Total inc. VAT	
23	07/17/2009 02:21 pm	$15.00	$15.00	view

Purchase list

Product	Amount	Total ex. VAT	Total inc. VAT
eZ Publish Advanced Content Management - Download: eZ Publish Advanced Content Management PDF	1	$15.00	$15.00

You can link to its "view" view (which takes one parameter: the user's ID) of such an order history module in another template, such as in a user's "My profile" page.

Other workflow uses

With the workflow extension as a starting point, here are a couple of the many other use cases for a pay-per-download solution that you could add with some extra coding:

- If you are selling software with product keys and have an existing product key generator, you could trigger this generator whenever someone purchases that product. Using the resulting key, you could create a new object with that key, and give the current user access to that object.

- Assign access to a particular section (via the assignment of a role or by moving the user to another group) to support the purchase of value-added website content (for content subscriptions and the like). Instead of buyers getting access to specific files, they receive access to entire areas of the site.

Limitations and other possibilities

Here are some limitations of the example extension and some hints on how to overcome them:

- To support multiple downloads per product, use an attribute of the "Related objects" datatype (instead of the "Related object" datatype we used), then in your workflow code (and all the templates that display details of an order), loop through all of the related objects.

- In our example, we've assumed that the buyer will complete payment via PayPal. You can

install different gateways, such as eZ Authorize, Beanstream (mugo.ca), or even accept offline payments.

- Neither the eZ PayPal extension nor the example pay-per-download extension change the order status after they've run. Therefore, by default the order statuses will all say "Pending". In the case of a pay-per-download situation, you have probably completed the entire buying loop and it might be useful to distinguish this as "Completed" or "Delivered". You can edit order statuses in the "Webshop > Order status" page of the "Administration Interface" and then employ the usage of eZOrder::modifyStatus() <http://pubsvn.ez.no/doxygen/4.1/html/classeZOrder.html#6c4dbbed0cab9fa3e6d6bc1f896f617d> in whatever shop-related extension you use.

Resources

You can download the example pay-per-download extension at:

`http://www.mugo.ca/Blog/Selling-Pay-Per-Download-Products`