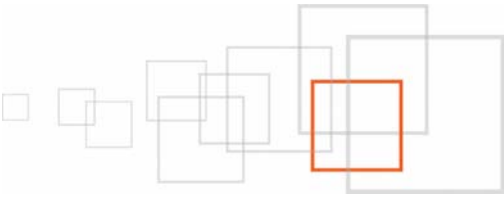
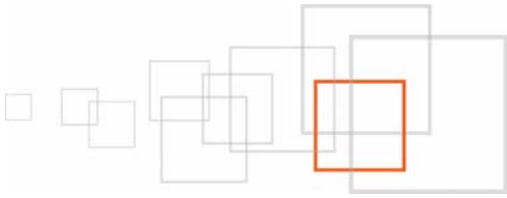
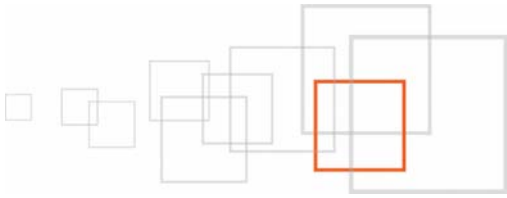


eZ Publish Cache In Details: expiry.php and ezcache.php

By Gilles Guirand
eZ Publish board member
CTO – Kaliop
@gandbox



Index



1 Goal description

This series of articles provides a rather advanced clarification on the operation of **the cache in eZ Publish**, with a particular focus on :

- the « **INI cache** » : PHP caching of the settings
- the « **template cache** » : compilation of *.tpl into *.php
- the « **view cache** » :caching of content stored in eZ Publish into an xHTML output (or others)
- the « **cache-block** », caching of the « **template-block** », bits of xHTML (or others) contained in the templates, and out of reach of the view cache (pagelayout.tpl, personalized module templates, etc.)
- the role of the **expiry.php** file
- the detailed operation of the **ezcache.php** script

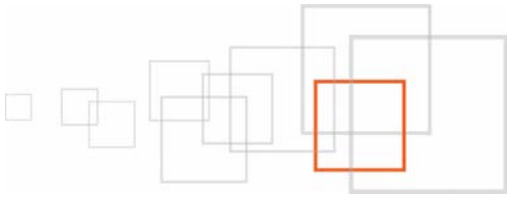
This series of articles will attempt to avoid explaining once again what is already comprehensible and has been fully described in other tutorials or official documentations, it aims :

- to explain how eZ Publish manages “low level” cache features that it offers : **this is the “documentation” aspect of the articles**
- To deduce **a possible impact, a possible good or bad practice** in the use and combination of the proposed parameters : **this is the “tutorial” aspect of the articles**

To all those who consider that eZ Publish cache management is complex and / or problematic, it is also necessary to consider the following :

- This tutorial exists (and this is a big difference), thank you for reading, critique, complete and tracing any questions or errors detected
- eZ Publish is resource intensive (including SQL) due to its way of storing data : the concept of class and the concept of EAV (Entity / Attribute / Value) is the price to pay to enjoy the magical concept of “content classes”, the functional stability or even the possible version upgrade starting from the origins of this CMS (stable & universal data model)
- Other “competitor” CMSs have a reputation to offer an easier to handle cache management simply because of the lack of actual cache management (usually a simple TTL, a delegation to Varnish, the absence of cache in authenticated mode...

Note : The mechanisms and examples described are based on version 4.5+ Entreprise of eZ Publish or eZ Publish 2011.x Community.



2 Introduction

eZ Publish is an infinitely flexible and powerful CMS that powers a wide variety of websites with low or high traffic. This power comes with a price : **resource consumption in general**, and SQL resource consumption in particular, as well as disk I/O ! All these mechanisms have to be controlled to size its architecture, or understand the impact of a parameter, of a cache-block directive or even a massive removal of cache (template, cache-block or view cache).

The previous articles detailed **templates compilation**, **view caches** and how to optimize **template-block caches** to ensure the sustainability of an eZ Publish project over time.

This article, that closes the series of four articles on eZ Publish caches, details the role of the **expiry.php** file and the way the **ezcache.php** script works.

3 Pre-requisites and target population

This series of articles is aimed mainly to:

- Developers / administrators with **an advanced knowledge of eZ Publish**
- Developers of high traffic websites or in search for a **performance boost**
- Developers who want to make peace with their server administrator / host
- Eco-conscious citizens, worried about wasted energy (or murdered Pandas!) caused by the rebuilding of view or template-block caches

WARNING : Reading this tutorial may be particularly painful and requires concentration and a long-lasting patience. It is advisable to read it in multiple sessions, equipped with a beer or a good bottle of wine for the wealthiest among us.

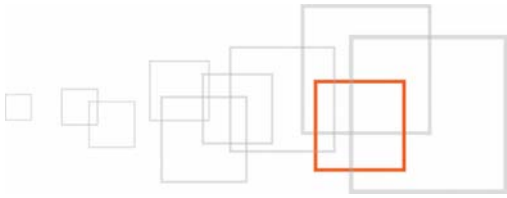
4 The transversal role of the expiry.php file

The **`{VarDir}/cache/expiry.php`** file was discussed throughout the articles. This file is used for a wide variety of things, for several caching mechanisms, when it comes to provide to PHP a basis for comparison with a **timestamp**.

For example, when the cache expires globally, rather than to invalidate (unlink, touch, update according to the filehandler) the whole set of cache files (which is impossible depending on the volume), the **expiry.php** provides an **up-to-date timestamp**, necessarily superior to all the **mtime** of the cache files produced so far.

The file consists of a simple key table:

```
<?php
$Timestamps = array (
'state-limitations' => 1321010927,
'user-info-cache' => 1323788789,
'content-view-cache' => 1326299075,
'class-identifier-cache' => 1326298969,
'global-template-block-cache' => 1325856140,
'content-tree-menu' => 1326298969,
```

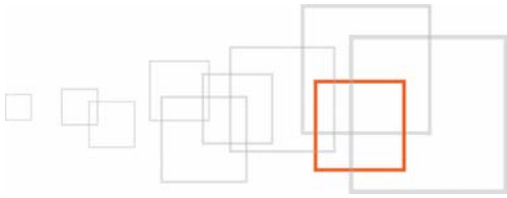


```
'image-manager-alias' => 1323788788,
'active-extensions-cache' => 1325668728,
'ts-translation-cache' => 1323788789,
'content-complex-viewmode-cache' => 1325780763,
'template-block-cache' => 1326298969,
'user-class-cache' => 1326298969,
'sort-key-cache' => 1326298969,
);
?>
```

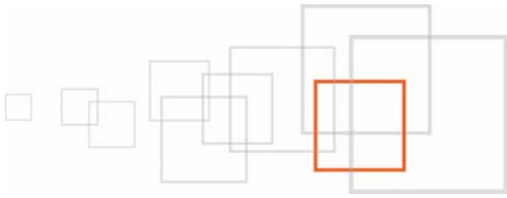
Some of these values have no real impact on performance, they are described briefly here for completeness.

Table of meanings of the couples of value / timestamp:

Key name	trigger(s)	Usage of the timestamp
state-limitations	***php bin/php/ezcache.php --clear-id=state_limitations*** ***php bin/php/ezcache.php --clear-tag=content*** **Editing of a status group of objects** **Reading of the limitations of a status group of objects**	**Expire la mise en cache d'un tableau à clé PHP décrivant la limitation sur un groupe d'objet d'état dans {VarDir}/cache/statelimitations_{hash}.php** **Expire the caching of a PHP key table describing the limitation on a set of status object in `{VarDir}/cache/statelimitations_{hash}.php`**
user-info-cache	***php bin/php/ezcache.php --clear-id=user_info_cache*** ***php bin/php/ezcache.php --clear-tag=user*** **Editing / Removal of a rule** **Editing / assigning / removal of a role** **Removal of a node**	**Globally expires the users cache (groups, rules, limitations by user)**
content-view-cache	***php bin/php/ezcache.php --clear-id=content*** ***php bin/php/ezcache.php --clear-tag=content*** **Exceeding of the CacheThreshold at the publication (250 nodes by**	**Globally expires the cache** **See the chapter describing the impact of this expiration on the performances**



	default) **Editing the roles, assignment of roles, creation / modification of content classes, assignment of sections** **See the chapter describing all the triggers**	
class-identifier-cache	***php bin/php/ezcache.php --clear-id=classid*** ***php bin/php/ezcache.php --clear-tag=content*** **Creation / modification of the content classes**	**Expires a PHP key table doing the link between the textual and numeric identifier of a class ("user" => "4") in `{VarDir}/cache/classidentifiers_{hash}.php`**
global-template-block-cache	***php bin/php/ezcache.php --clear-id=template-block*** ***php bin/php/ezcache.php --clear-tag=template*** ***php bin/php/ezcache.php --clear-tag=content***	**Expires all the template-blocks caches (the cache-blocks), including those containing the subtree_expiry directive**
content-tree-menu	***php bin/php/ezcache.php --clear-all*** ***php bin/php/ezcache.php --clear-tag=content*** ***php bin/php/ezcache.php --clear-id=content_tree_menu***	**Allows the Back Office to cache the browser AJAX requests to the content browser** **This value is available in the templates using the `{fetch('content','content_tree_menu_expiry')}` function**
image-manager-alias	***php bin/php/ezcache.php --clear-id=imagealias*** ***php bin/php/ezcache.php --clear-tag=image***	**Expires the images alias list defined in image.ini / [AliasSettings]**
active-extensions-cache	***php bin/php/ezcache.php --clear-id=active_extensions*** ***php bin/php/ezcache.php --clear-tag=ini***	**Expires the caching of a PHP key table doing the list of the activated extensions for each site_acces in `var/cache/active_extensio`**



		`ns_{hash}.php`
ts-translation-cache	***php bin/php/ezcache.php --clear-id=translation*** ***php bin/php/ezcache.php --clear-tag=i18n***	**Globally expires the translation PHP key tables, stored according to the site_access in `{VarDir}/cache/translation/{hash}/{language}/{hash}.php`**
content-complex-viewmode-cache	**Content publication**	**Expires the view caches dependant to a viewmode too complex to be linked to individual files (sitemap for example), according to the parameter : site.ini `[ComplexDisplayViewModes]`**
template-block-cache	***php bin/php/ezcache.php --clear-tag=clear-all*** **Content publication** **Editing the roles, assignment of roles, creation / modification of content classes, assignment of section**	**Expires the whole template-blocks cache (the cache-blocks), except those containing a subtree_expiry directive**
user-class-cache	**Edition of a role** **Creation / modification of a class**	**Expires the session cache determining if the list of classes that the current user can create should be updated or not** **This class list is available with the `fetch( 'content', 'can_instantiate_class_list' )` function**
sort-key-cache	***php bin/php/ezcache.php --clear-id=sortkey*** ***php bin/php/ezcache.php --clear-tag=content*** **Creation / modification of a class**	**Globally expires the datatype PHP key table linked to each sort for each datatype (“ezstring” => “string”) in `{VarDir}/cache/sortkey_{hash}.php`**
ezpRestRouteApcCacheKey	***php bin/php/ezcache.php --clear-id=rest-routes*** ***php bin/php/ezcache.php***	**Related to the REST API and the removal of APC routes, not tested**

	--clear-tag=rest	

5 The ezcache.php script

This table summarizes the list of available parameters for the **ezcache.php** script and their impact on the system. We must distinguish 3 types of parameters :

- **--clear-all :** this parameter targets every kind of cache
- **--clear-tag :** this parameter a set of caches (ID list), thematically groupes in the same tag
- **--clear-id :** this parameter targets a single type of cache

You can add other parameters when calling the script

- **--purge** : concernés requires a purge of the files, with a more or less real impact depending on the ID. The table below describes the impact of the --purge directive for each of these IDs
- **--expiry :** allows to define an expiry date to be consider, 'now', '-2 days', 'last monday', only for ID-related functions
- **--s --siteaccess :** Used to force the use of a site_access to use the relevant settings, if not given the default site_access is used
- **--iteration-sleep :** not tested
- **--iteration-max :** not tested

Other parameters are dedicated to assistance :

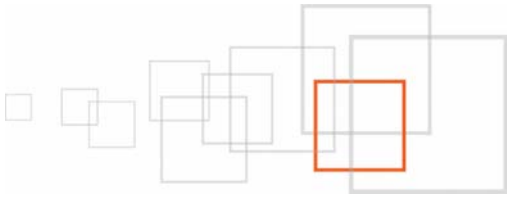
- **--help** : displays help with the list of parameters and their meanings (less detailled than in the document)
- **--list-ids** : displays the list of usable IDs
- **--list-tags** : displays the list of usable TAGs

When calling a cache expiry by the **ezcache.php** command, the PHP array of the **kernel/classes/ezcache.php** class is used to define what type of action to run.

```
array( 'name' => ezpI18n::tr( 'kernel/cache', 'Text to image cache' ),
       'id' => 'texttoimage',
       'tag' => array( 'template' ),
       'enabled' => $textToImageIni->variable( 'ImageSettings', 'UseCache' ) ==
'enabled',
       'path' => $textToImageIni->variable( 'PathSettings', 'CacheDir' ),
       'function' => array( 'eZCache', 'clearTextToImageCache' ),
      'purge-function' => array( 'eZCache', 'purgeTextToImageCache' ),
      'is-clustered' => true )
```

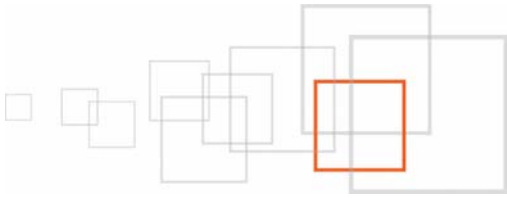
These parameters are used to define the type of action to take. Here is an excerpt of the relevant PHP code (**kernel/classes/ezcache.php**), which is relatively understandable by reading his algorithm:

```
// Recovering any function
// - If --purge and 'purge-function' exists
// - Or if --purge and 'function' doesn't exists
if ( $purge && isset( $cacheItem['purge-function'] ) )
```



```
      $functionName = 'purge-function';
else if ( !$purge && isset( $cacheItem['function'] ) )
       $functionName = 'function';

// Then, if a function exists
if ( $functionName )
{
      // ... executes the function declared in ‘function’
      call_user_func_array( $function, array( $cacheItem ) );
}
else
{
      // If ‘is-clustered’ is true, we force the recursive removal in cluster mode
      // touch, update SQL... depending on the filehandler defined in file.ini
       if ( $isClustered )
      {
             $fileHandler = eZClusterFileHandler::instance( $cachePath );
             if ( $purge )
             $fileHandler->purge( $reporter, $iterationSleep, $iterationMax, $expiry );
             else
                    $fileHandler->delete();
                     return;
      }
      // ... Recursively removes (unlink) the path defined in 'path'
      eZDir::recursiveDelete( $cachePath );
}
```

Many bugs appeared in version eZ 4.x until version 4.5 (the informations of the table below are for eZ Publish 2012.x) that do not systematically defined value for ‘**function-purge**’. This lack os definition caused an unpredictable effect when using **--purge**, ie **no action** (Reading again the algorithm allows us to understand that if --purge is invoked and that only the ‘function’ parameter is present, then no specific PHP function is called).



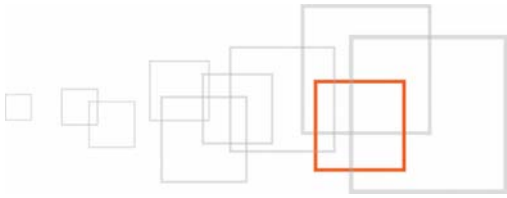
5.1 List of roles, ID and Tags by default

Table of ID / TAG of the ezcache.php command / impact of --purge

TAG	ID
content	content / classid / sortey / urlalias / rss_cache / content_tree_menu / state_limiations / template-block / ezjscore-packer
template	template / template-block / template-override / texttoimage / design_base / ezjscore-packer
ini	ini / global_ini / active_extensions
user	user_info_cache
I18n	translation / chartrans
codepage	codepage
image	imagealias
rest	rest / rest-routes

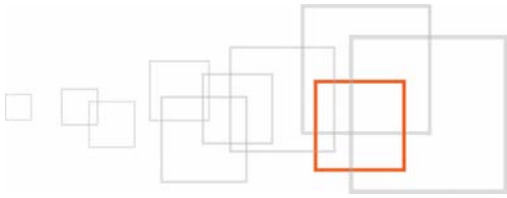
ID	associated tag(s)	Meaning / impact
content	content	Globally expires the view cache Update of the timestamp of 'content-view-cache' in `{varDir}/cache/expiry.php` See chapter describing the impact of this expiry on performances --purge ? Physically removes this cache by recursive UNLINK of `{varDir}/cache/content` or DELETE SQL if an eZDB / eZDFS filehandler is used
classid	content	Expires a PHP key table doing the link between a

		textual and numeric class identifier : (“user” => “4“) in `{VarDir}/cache/classidentifiers_{hash}.php` `{VarDir}/cache/classattributeidentifiers_{hash}.php` **Removes these 2 files --purge ? useless, no additional impacts**
sortkey	**content**	**Globally expires the PHP key table of the datatypes linked to each datatype (“ezstring” => “string”) in `{VarDir}/cache/sortkey_{hash}.php`** **Removes this file** **--purge ? useless, no additional impacts**
urlalias	**content**	**Globally expires the ‘wildcard’ cache** **Invalidates this cache by recursive UNLINK of `{varDir}/wildcard`, TOUCH if eZFS2 is used or UPDATE SQL eZDB / eZDFS are used** **--purge ?** **Physically removes this cache by recursive UNLINK of `{varDir}/wildcard` or DELETE SQL if an eZDB / eZDFS filehandler is used**
rss_cache	**content**	**Globally expires the ‘wildcard’ cache** **Invalidates this cache by recursive UNLINK of `{varDir}/wildcard`, TOUCH if eZFS2 is used or UPDATE SQL eZDB / eZDFS are used** **--purge ?** **Physically removes this**

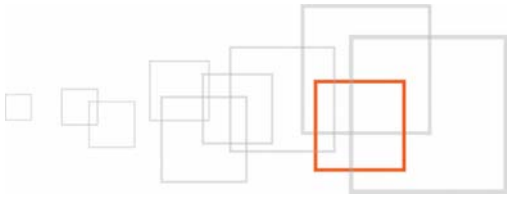


		cache by recursive UNLINK of `{varDir}/wildcard` or DELETE SQL if an eZDB / eZDFS filehandler is used
rss_cache	content	Globally invalidates RSS cache Invalidates this cache by recursive UNLINK of `{varDir}/rss`, TOUCH eZFS2 is used or UPDATE SQL if eZDB / eZDFS are used --purge ? Physically removes this cache by recursive UNLINK of `{varDir}/rss` or DELETE SQL if an eZDB / eZDFS filehandler is used
content_tree_menu	content	Allows the Back Office to cache the browser AJAX requests to the content browser Updates the timestamp of the 'content-tree-menu' in `{varDir}/cache/expiry.php` This value is available in templates with the `{fetch('content','content_tree_menu_expiry')}` function --purge ? useless, no additional impacts
state_limitations	content	Expires the caching of a PHP key table describing the limitation on a group of status objects in `{VarDir}/cache/statelimitations_{hash}.php` Removes this file by UNLINK --purge ? useless, no additional impacts

template	**template**	**Globally expires the compiled version of templates** **Physically removes this cache by recursive UNLINK of `{varDir}/cache/template/`** **--purge ? useless, no additional impacts**
template-block	**template content**	**Globally expires the template-block cache** **Update the timestamp of 'global-template-block-cache' in `{varDir}/cache/expiry.php`** **--purge ?** **no update of expiry.php** **Recursively removes `{varDir}/cache/template-block`** **See chapter describing the impact of this expiry on performances**
template-override	**template**	**Globally expires the cache of definition of overrides of templates** **Invalidates this cache by recursive UNLINK of `{varDir}/cache/override/` that contains the key tables that links the template overrides** **Deletes the array in $GLOBALS** **--purge ?** **Same thing, but doesn't remove the array in $GLOBALS, which seems to be a lack of consistency for a 'purge-function'. Moreover,**

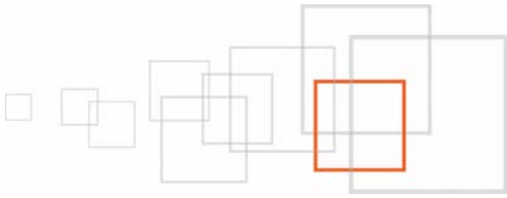


		this removal is not helpful in any way, therefore this error doesn't have an impact
texttoimage	template	Globally expires the texttoimage cache. Not tested, very marginal feature. Invalidates this cache by recursive UNLINK of `var/cache/texttoimage`, TOUCH if eZFS2 is used. --purge ? Physically removes this cache by recursive UNLINK of `var/cache/texttoimage`. The `var/cache/texttoimage` directory is defined by the parameter `texttoimage.ini / [PathSettings] / CacheDir`
design_base	template	Globally expires the design/ directories cache Invalidates this cache by recursive UNLINK of the `{varDir}/cache/designbase_*` files, TOUCH if eZFS2 is used or UPDATE SQL if eZDB / eZDFS are used --purge ? useless, no additional impacts
ezjscore-packer	content template	Globally expires the ezjscore cache (scripts & CSS) Invalidates this cache by recursive UNLINK of the `{varDir}/public`, TOUCH if eZFS2 is used or UPDATE SQL if eZDB / eZDFS are used --purge ? Physically removes this cache by recursive UNLINK of `{varDir}/public` or



		DELETE SQL if eZDB / eZDFS filehandlers are used
global_ini	**ini**	**Globally expires the INI cache** **Invalidates this cache by recursive UNLINK of `var/cache/ini/` that contains the INI caches (PHP key arrays), whatever the filehandler is** **--purge ? useless, no additional impacts**
active_extensions	**ini**	**Globally expires the active extensions per site_access cache** **Invalidates this cache by recursive UNLINK of the `var/cache/active_extensions_{hash}.php` files** **Update of the timestamp of 'active-extensions-cache' in `{varDir}/cache/expiry.php`** **--purge ? useless, no additional impacts**
ini	**ini**	**Doesn't do anything, or at least tries to remove recursively a directory that doesn't exist (for the record : `{varDir}/ini`). The 'var/cache/ini/' directory is hard writen in `lib/ezutils/classes/ezini.php`** **Use global_ini to remove the INI cache**
user_info_cache	**user**	**Globally expires the cache of user rules** **Update of the timestamp of 'user-info-cache' in {varDir}/cache/expiry.php**

		--purge ? **Physically removes this cache by recursive UNLINK of {varDir}/user-info or DELETE SQL if eZDB / eZDFS filehandlers are used**
translation	**i18n**	**Globally expires the translation cache** **Update of the timestamp of 'ts-translation-cache' in {varDir}/cache/expiry.php** **--purge ?** **Physically removes this cache by recursive UNLINK of {varDir}/translation**
chartrans	**i18n**	**Globally expires the character transformation cache (for example from uppercase to lowercase)** **Physically removes this cache by recursive UNLINK of `{varDir}/trans`, whatever the filehandler is** **--purge ? useless, no additional impacts**
codepage	**codepage**	**Globally expires the *codepages* transformation cache (links between encodings)** **Physically removes this cache by recursive UNLINK of {varDir}/codepages, whatever the filehandler is** **--purge ? useless, no additional impacts**
imagealias	**image**	**Globally expires the image aliases cache** **Update of the timestamp of the 'image-manager-alias' in `{varDir}/cache/expiry.php`**



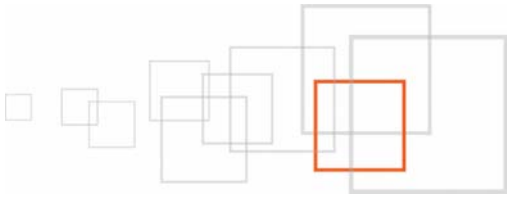
		--purge ? **Adds to the previous operation the removal in database of the 'alias' in all the attributes of the 'ezimage' datatype**
rest	**rest**	**Globally expires the REST API cache** **Invalidates this cache by recursive UNLINK of the `{varDir}/rest`, TOUCH if eZFS2 is used or UPDATE SQL if eZDB / eZDFS are used** **--purge ?** **Physically removes this cache by recursive UNLINK of `{varDir}/rest` or DELETE SQL if eZDB / eZDFS filehandlers are used**
rest-routes	**rest**	**Globally expires the APC cache of the routes of the REST API (experimental, not tested)** **Update of the timestamp of 'ezpRestRouteApcCacheKey' in `{varDir}/cache/expiry.php`**

5.2 Add their own ID and Tags in an extension

When installing or developping a new extension, or following a kernel update, it is possible to obtain new cache expiry identifiers, related to this extension or kernel update. The new idientifiers are declared **outside the ezcache.php file** (which potentially could be done for all identifiers) in the **settings/site.ini** configuration file (for the kernel) or **extension/ {extension_name}/settings/site.ini.append.php**

On the version of eZ Publish currently used for this article (eZ 2012.x), we can find **two illustration statements outside the ezcache.php file**:

- The cache expiry identifiers for the **REST API (rest and rest-routes)**, declared in **settings/site.ini**, because the **REST API** is not an extension but a kernel evolution
- The cache expiry identifiers for **eZJSCore (ezjscore-packer)**, declared in



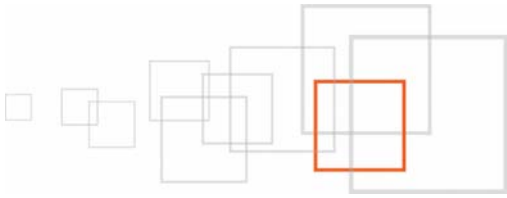
extension/ezjscore/settings/site.ini.append.php

The **site.ini** file thus produces both a documentation and a usage example of the parameters:

```
[Cache]
# Array with cache items to extend ezcache
CacheItems[]

# Example of use:
#CacheItems[]=custom
#
#[Cache_custom]
#
# [optional] Name of the cache item, key (custom) is used camel-cased if not set
#name=Custom cache
# [optional] Id of cache item, to be used from command line, key (custom) is used if
not set
#id=custom
# [optional] If cache item should be cleared using cluster instead of plain file
handler, def: false
#isClustered=true
# [optional] tags that will trigger clearing of this item, default: empty array
#tags[]=content
#tags[]=custom
# [optional] If cache uses eZExpiryHandler, then this is the key to get expiry time
#expiryKey=global-custom-cache
# [optional] (bool) If this cache is enabled or not, default: true
#enabled=false
# [optional] Path where cache is stored, either directory or file if class/purgeClass
# variables are unset, relative to current cache directory is assumed
#path=custom_cache.php
# [optional] custom cache clear function "<class>::clearCache()" to be called
#class=eZSomeClassName
# [optional] custom cache purge function "<class>::purgeCache()" to be called
#purgeClass=eZSomeClassName

CacheItems[]=rest
CacheItems[]=restRoutes

[Cache_rest]
name=REST Application cache
id=rest
tags[]=content
tags[]=rest
path=rest
isClustered=true

[Cache_restRoutes]
name=REST Routes memory cache
id=rest-routes
tags[]=rest
class=ezpRestRoutesCacheClear
purgeClass=ezpRestRoutesCacheClear
```

The **extension/ezjscore/settings/site.ini.append.php** file provides an example of usage in an extension context:

```
# Cache item entry (for eZ Publish 4.3 and up)
```



```
[Cache]
CacheItems[]=ezjscore

[Cache_ezjscore]
name=eZJSCore Public Packer cache
id=ezjscore-packer
tags[]=content
tags[]=template
path=public
isClustered=true
```

6 Conclusion

An eZ Publish website changes over time in terms of functionality, volume, attendance and therefore activity and behavior of its cache. The low-level understanding of the **expiry.php** file and **a judicious use of the ezcache.php script** are two important prerequisite to master live deployments and diagnose possible massive caches expiry that can impact the proper functioning of a website.

7 About the author : Gilles Guirand

Gilles Guirand (CTO Kaliop – eZ Publish Platinum partner) combines 12 years of experience in Web development, 5 of which dedicated specifically on the deployment of large national projects using eZ Publish. He is today a regular conference speaker, an active contributor (Award 2011) and a board member of eZ Publish community.

8 License choice

9 Appendix

Many thanks to all contributors who participate in the development of these tutorials :

- Kaliop (time & case studies)
- Samuel Bouic (EN translation)
- eZ engineering (reviewing)
- Charlotte Langlois (formatting)