# Magento
# Blog Extension

TECHNICAL DOCUMENTATION

by

evozon

# Magento Blog Extension Contents

CHAPTER 1

# 1. Installing the extension

You can install this extension by downloading the Magento source archive from our website, and use your Magento Connect account to upload it. Don`t forget to flush the Magento cash. Make sure to log out when you`re done.

CHAPTER 2

# 2. Blog Category

## 2.1 About

Magento Category can easily be transformed into a blog category and then it displays Evozon_Blog_Model_Post entities instead of products.

## 2.2 Adding a new layout for category view

While having a category set to act as a blog page (Catalog->Manage Categories->Custom Design tab->Is Blog Category? attribute-> Yes), the layout applied can be changed (rather than using the one set as Default Layout in System->Configuration->Evozon Blog->General Configuration->Layout Settings).

In order to make changes on the category content, you should use the *posts.category* node (for category modifications) and *post_list* node (for posts listing content) by editing the *Custom Layout Update* field.

Ifyou want to change the layout of the columns(left/right/etc), edit your layout xml *posts.tags, posts.categories, posts.comments.recent, posts.recent, posts.archive* and *posts.rss* under given references (left or right).

## 2.3 Changing the *featured blog posts* block position in a non-blog category

As a developer (or back-end user), you have full control of the layout by editing the *Custom Layout Update* block (Catalog->Manage Categories->Custom Design tab). While having a non-blog category with the *Show Featured Blog Posts* attribute set to “Yes”, the user will see the articles selected in the *Blog Posts* tab under the products list.

To move the *featured block posts* block in the left column (if your selected layout allows you to), add this to the textarea:

```
<reference name="left">
      <block type="evozon_blog/catalog_category_posts"
name="evozon_blog_post_category_articles_left"
template="evozon/blog/catalog/category/posts/columns.phtml"/>
</reference>
```

To change the position of the block in the right column, copy/paste the following.

```
<reference name="right">
      <block type="evozon_blog/catalog_category_posts"
name="evozon_blog_post_category_articles_right"
template="evozon/blog/catalog/category/posts/columns.phtml"/>
</reference>
```

Also, feel free to change the template if you decide to use a custom view of the related posts.

CHAPTER 3

# 3. Posts

## 3.1 About

Evozon Blog allows the user to create blog articles (*posts*) the same way the Mage Catalog creates Products. On setup, the *evozon_blog_post* entity type creates the required tables (attribute table - *evozon_blog_eav_attribute* and EAV required tables).

In order to reuse the Magento code and to offer developers a familiar experience while working with Evozon's Blog extension, and to also make use of some of the products` behaviour (mainly the EAV benefits and catalog/website relations), the *evozon_blog_post* entity extends from product in the following way: the model - *Mage_Catalog_Model_Product*, the resource - *Mage_Catalog_Model_Resource_Abstract* and the collection - *Mage_Catalog_Model_Resource_Product_Collection*.

## 3.2 Adding new attributes

The blog posts entity can be managed as a product. In order to add a new attribute, create a new installer script and set the type of *Evozon_Blog_Model_Post::ENTITY.*

## 3.3 Changing sorting attributes

Similar to managing a product, you can add attributes with the property of *used_for_sort_by* set to "1".
In order to see which attributes are currently used for sort by, you can use the public function *Mage::getSingletonl('evozon_blog/config')->getAttributesUsedForSortBy()* or by

calling out for *Mage::getSingleton('evozon_blog/config')->getAttributeUsedForSortByArray()*.

## 3.4 Adding new gallery templates

On top of the currently available gallery templates - slideshow, thumbnails -, a developer has the freedom to add even more templates to the gallery widget.

The following elements will be required:

1. a new template file.
2. the selected js/css library.

The template file (ex: [..]/NEW_TEMPLATE.phtml) will use the *getSelectedImages()* to display the images selected for specific type and *getGalleryId()* to uniqely identify it.

In your project`s **widget.xml**, add a new child to the ***<widgets>*** node, having the following path:

```
<evozon_blog_post_gallery>
  <parameters>
     <display>
        <values>
           <NEW_GALLERY_TYPE>
              <label>NEW_GALLERY_TYPE_LABEL</label>
          <value>NEW_GALLERY_TYPE_TEMPLATE_PATH</label>
           </NEW_GALLERY_TYPE>
        </values>
     </display>
  </parameters>
</evozon_blog_post_gallery>
```

Feel free to add other parameters if your new gallery library requires this. The values selected can further be accessed from the template by using *$this->getData('NEW_GALLERY_TYPE_PARAMETER')*.

The *Evozon_Blog_Block_Adminhtml_Post_Edit_Gallery_Images* block will use your template to apply your new gallery library.

Add the selected css/js to your theme and to the *<evozon_blog_post_view></evozon_blog_post_view>* node in your layout xml or directly in the template file (in order to avoid loading it on all the pages).

## 3.5 Adding new template for recent posts

If you have decided not to alter the extisting templates in the project`s theme (Extended and Minimal), you can add new templates/types to render the *Evozon_Blog_Block_Post_List_Recent_Posts* block.

The new template can be added to be used for both the widget and the block.

*1. Adding it for the widget*

In your project`s **widget.xml**, add a new child to the ***<widgets>*** node, having the following path:

```
<evozon_blog_post_recent>
  <parameters>
     <template>
        <values>
           <NEW_WIDGET_TYPE>
              <label>NEW_WIDGET_TYPE_LABEL</label>
          <value>NEW_WIDGET_TYPE_TEMPLATE_PATH</label>
           </NEW_WIDGET_TYPE>
         </values>
     </template>
  </parameters>
</evozon_blog_post_recent>
```

*$this->getRecentPosts()* will be used in the template file to get the post entities.

*2. Adding it for the block*

As it can be seen in Evozon Blog`s extension **system.xml**, the source_model for the template`s field is Evozon_Blog_Model_Adminhtml_System_Config_Source_Recent_Posts_Template.

In order to add customized templates, create a source model to extend *Evozon_Blog_Model_Adminhtml_System_Config_Source_Recent_Posts_Template*, where the *toOptionArray()* function can be used to add other templates to the *parent::toOptionArray()* array (the original one).

In your project`s **system.xml**, add a new child to the **<sections>** node as following:

```
  <evozon_blog_post>
     <groups>
        <recent>
           <fields>
              <template>
                  <source_model>YOUR_SOURCE_MODEL<source_model>
              </template>
           </fields>
         </recent>
      </groups>
  </evozon_blog_post>
```

If other parameters are required for the customized template, feel free to add them. In order to access their values, use the function *getConfigData($key)* from the Evozon_Blog_Block_Post_List_Recent_Posts block in the created template.

### 3.6 Adding new layout for specific post

Similar to changing the product view template, you can set your own template to render a post. Change the Design package, the page layout and individual segments in the layout by using the xml.

For example, if you wish to give a different *content* child node to the page layout, use the reference name "*evozon_blog_post"* to set a different template:

```
<reference name="evozon_blog_post">
      <action method="setTemplate">
            <template>YOUR_TEMPLATE</template>
      </action>
</reference>
```

The changes made for altering the design/layout are only active for the <evozon_blog_post_view> action.

CHAPTER 4

# 4. Archive

### 4.1 About

Archive has been designed as a tool to filter the blog posts by periods and intervals. It displays the archive on the following "periods": *this Week,* and *by years* (which will further display the posts *by month*). If there are no posts available, the period won`t be shown.

*Evozon_Blog_Block_Post_Archive_Block* calls for the resource model *Evozon_Blog_Model_Resource_Post_Archive* to fetch archive collection (period and count of posts) and then to fetch the monthly collection.

Basically, depending on the conditions set on the resource, different parts of an SQL are joined in the function *getArchiveSelects()* and then the database call is made to fetch all the results.

### 4.2 Adding new periods to the archive

The directing class is *Evozon_Blog_Model_Resource_Post_Archive*, where in the getArchiveCollection(), a new archive condition (month) can be added using the protected function *_addArchiveConditions()*.

All existing conditions must extend the abstract class *Evozon_Blog_Model_Resource_Post_Archive_Condition_Abstract*. In this class, the base of the SQL parts is created: the filtering on the posts collection *getPostFilteredSelect()*, that will return an SQL with entity_id and publish_date. This data is used to group the post counts by periods.

In the abstract class there are four helping functions: *getDateFormat()*, *getWhere()*, *getResultLabel()*, *getResultUrl()*:

- Use *getDateFormat()* to set the date format used in the mysql function DATE_FORMAT in order to generate intervals;
- Use *getWhere()* to set the dividing condition;
- Use *getResultLabel()* to set the label for the period, as displayed on the frontend;
- Use *getResultUrl()* to set the url it will respond to.

For example, in order to add a new "period" (such as Past Month), a new *Evozon_Blog_Model_Resource_Post_Archive_Condition_Abstract* will be created, having the following data set:

```
protected function getDateFormat(){
      return '%m';
}


protected function getWhere(){
      return 't.publish_date > DATE_SUB(now(), INTERVAL 1 MONTH)';
}


protected function getResultLabel(){
      return '"Past month"';
}


protected function getResultUrl(){
      return '"month"';
}
```

CHAPTER 5

# 5. Search

## 5.1 About

The search feature implemented in Evozon's Blog Extension comes with a pack of specifications:

a) a filtering block added at the top of the navigation layer (the "Search In" block), which allows the user to chose which results to see (blog posts or products);
b) the products related to blog posts that match the query will also be added to the products result;
c) more engines can be added (Solr, ElasticSearch, etc) and used to search through post collection.

In order to allow that, two classes from the Magento core have been **overwritten**:

1. Mage_CatalogSearch_Block_Layer for Community Edition or Enterprise_Search_Block_CatalogSearch_Layer for Enterprise Edition - in order to attach the Search In block and optimize the work with the collections;
2. Mage_CatalogSearch_Model_Layer - in order to make changes in the prepared product collection. The product collection will be changed only if the search feature of the Evozon's Blog Extension is Enabled.

## 5.2 Adding a new search engine

A developer has the posibility of adding new search engines that must have their own indexer and collection.

For this, you`ll have to add a new child under the node <config> in your xml file:

```
<search>
	<engines>
		<ENGINE_NAME>
			<engine>
				<label>ENGINE_LABEL</label>
				<model>ENGINE_MODEL</model>
				<resource>ENGINE_RESOURCE</resource>
			</engine>
			<collection>
				<resource>SEARCH_POSTS_COLLECTION</resource>
			</collection>
			<indexer>
				<model>ENGINE_INDEXER_MODEL</model>
				<resource>ENGINE_INDEXER_RESOURCE</resource
			</indexer>
		</ENGINE_NAME>
	</engines>
</search>
```

ENGINE_RESOURCE must extend *Evozon_Blog_Model_Resource_Search_Engine_Abstract*.

CHAPTER 6

# 6. Spam

## 6.1 About

The already existing services can be found in the *spam.xml* configuration file. This file is read by *Evozon_Blog_Model_Spam_Config*.

## 6.2 Adding a new spam service

A developer has the possibility to add new spam services in order to detect and remove spam messages.

For this, you`ll have to add a new child under the node <config> in your xml file:

```
<crawler>
	<services>
		<SERVICE_NAME>
			<!--- you should also specify the label and the model
	used  →
			<label>SERVICE_LABEL</label>
			<model>SERVICE_MODEL</model>
		</SERVICE_NAME>
	</services>
</crawler>
```

*SERVICE_MODEL* must extend the abstract class *Evozon_Blog_Model_Spam_Service_Abstract* and implement the interface *Evozon_Blog_Model_Spam_Service_Interface.*

Follow your spam checker`s documentation in order to implement the required functions by the interface.

CHAPTER 7

# 7. Restrictions

## 7.1 About

The restriction feature allows to set restrictions on *Action* or on *Action and Component* (this is the "rule" part of the restriction) for different *group of customers* ( - the "container" part). Restriction rules are set on post visibility on *listing* and *view*ing an article. The restriction container will apply the rules in the *validate()* function.

### 7.2 Adding a new restriction container

The main container has a collection of containers and is located in Evozon_Blog_Block_Adminhtml_Post_Edit_Tab_Restrictions_Renderer_Container_Simple block. Your new container should be in the array of *getNewChildSelectOptions()*.

The newly added model which contains the rule will extend Evozon_Blog_Model_Restriction_Container_Abstract_Container and will implement Evozon_Blog_Model_Restriction_Container_Interface_Container.
Its resolver function (described further) will be mentioned and used to set data.

The *$_renderer* will set the block rendering this container. This rendering block keeps the type of restrictions which will be rendered further. It has to extend Evozon_Blog_Block_Adminhtml_Post_Edit_Tab_Restrictions_Renderer_Abstract_Container and to implement Evozon_Blog_Block_Adminhtml_Post_Edit_Tab_Restrictions_Renderer_Interface_Container

The *$_commentRenderer* value indicates the comment renderer class and it must extend Evozon_Blog_Block_Adminhtml_Post_Edit_Tab_Restrictions_Renderer_Abstract_Comment. Make sure to set proper $_renderer and $_commentRenderer values.

The next step is creating the $_commentRenderer container block, which is displayed in Restrictions tab when selected in the first dropdown. Create the above mentioned class and use *_getCommentOptions()* to set the options for the operator (is, is not, has, has not, etc) and the list of items related to the rule (the available implementation is for *user groups*). Use *getComment()* to return the rule comment.

In order to apply the rule, each newly introduced element (required by your container) must have a *resolver* that manages it. The resolvers are located in *Evozon/Blog/Model/Restriction/Util/Resolver* folder and they must implement the *Evozon_Blog_Model_Restriction_Util_Interface_DataRequestResolver* interface. Following the required functions, set the validation and data retrieval on this model.

CHAPTER 8

# 8. Validation

### 8.1 About

Evozon's Blog Extension accepts user input (such as tags and comments - from both admin and user) which requires validation. The validation on each model is a variation of *Zend_Filter_Input*, which allows to customize each field individually as well as setting general filters and options.

The already existing validation rules can be found in the *validation.xml* configuration file. This file is processed by *Evozon_Blog_Model_Filter_Config*.

## 8.2 Adding a new validation model

A developer has the posibility to add new validation models in order to filter and validate the fields.

For this, you`ll have to add a new child under the node <config> in your xml file:

```
<validation>
	<tags>
		<VALIDATION_NAME>
			<!--- you should also specify the label and the model
	used  →
			<label>VALIDATION_LABEL</label>
			<model>VALIDATION_MODEL</model>
		</VALIDATION_NAME>
	</tags>
</validation>
```

*VALIDATION_MODEL* must implement the interface *Evozon_Blog_Model_Filter_Interface.* *VALIDATION_MODEL* will have the input data (the object to be validated/filtered/sanitized) given by the factory class *Evozon_Blog_Model_Filter_Factory* (which extends *Zend_Filter_Input*).

*Evozon_Blog_Model_Filter_Interface* has three functions:

1. *getOptions()* - add here an array that will contain options for the entire group of data;
2. *getFilters()* - configurate filters (such as stringtrims, striptags, etc);
3. *getValidators*() - will have to contain validators for each input (if there are any particularities).

Use this guide on *Zend_Filter_Input* to help you set the options, filters and validators.

# CHAPTER 9

# 9. Uninstalling the extension

The Evozon's Blog extension doesn`t create dependencies to itself, so just deactivating the extension should do the work.