pimcore

Documentation

Pimcore Documentation

Table of Contents

Templates (Views)

As mentioned already before, Pimcore uses Zend_View as its template engine, and the standard template language is PHP.

The Pimcore implementation of Zend_View offers special methods to increase the usability:

Method	Description
inc	Use this function to directly include a document
template	Use this method to include a template
cache	In template caching
translate	i18n / translations
glossary	Glossary

Additionally you can use the Zend_View helpers which are shipped with ZF. There are some really cool helpers which are really useful when used in combination with Pimcore.

Some Examples

Method	Description
action	http://framework.zend.com/manual/en/zend.view.helpers.html#zend.view.helpers.initial.action
headMeta	http://framework.zend.com/manual/en/zend.view.helpers.html#zend.view.helpers.initial.headmeta
headTitle	http://framework.zend.com/manual/en/zend.view.helpers.html#zend.view.helpers.initial.headtitle
translate	http://framework.zend.com/manual/en/zend.view.helpers.html#zend.view.helpers.initial.translate

You can use your own custom Zend_View helpers, or create some new one to make your life easier.

There are some properties which are automatic available in the view:

Name	Type	Description
editmode	boolean	Is true if you are in editmode (admin), false if you are on the website
controller	Pimcore_Controller_Action_Frontend	A reference to the controller
document	Document	Reference to the current document object you can directly access the properties of the document in the view (eg. $this?document?getTitle();)

Editables (Placeholders for content)

Pimcore offers a basic set of placeholders which can be placed directly into the template. In editmode they appear as an editable widget, where you can put your content in. While in frontend-mode the content is directly embedded into the HTML.

There is a standard scheme for how to call the editables. The first argument is always the name of the element (as string), the second argument is an array with multiple options (configurations) in it.
Because most of the elements are based directly on Ext.form elements, you can also pass configurations directly to the Ext components (see API reference of Ext)

Click here to get a detailed overview about the editables.

Example

```
<!-- creates a input in editmode (admin) and directly outputs the text in frontend -->
<h1><?= $this->input("headline", array("width" => 540)); ?></h1>

<!-- advances template -->
<?php $this->layout()->setLayout('standard'); ?>


<h1><?= $this->input("headline", array("width" => 540)); ?></h1>
<h1><?= $this->numeric("number", array("width" => 540)); ?></h1>

<?php while ($this->block("contentblock")->enumerate()) { ?>
    <?php if($this->editmode) { ?>
        <?= $this->select("blocktype",array(
            "store" => array(
                array("wysiwyg", "WYSIWYG"),
                array("contentimages", "WYSIWYG with images"),
                array("video", "Video")
            ),
            "onchange" => "editWindow.reload.bind(editWindow)"
        )); ?>
    <?php } ?>

    <?php if(!$this->select("blocktype")->isEmpty()) {
        $this->template("content/blocks/".$this->select("blocktype")->getData().".php");
    } ?>
<?php } ?>
```

Editables

The editables are placeholders in the templates, which are input widgets in the admin (editmode) and output the content in frontend mode.

- Area (since 1.4.3)
- Areablock (since 1.3.2)
- Block
- Checkbox
- Date
- Href (1 to 1 Relation)
- Image
- Input
- Link
- Multihref (since 1.4.2)
- Multiselect

- Numeric
- Renderlet
- Select
- Snippet (embed)
- Table
- Textarea
- Video
- WYSIWYG

General

Most of the editables use ExtJS widgets, these editables can be also configured with options of the underlying ExtJS widget.

For example:

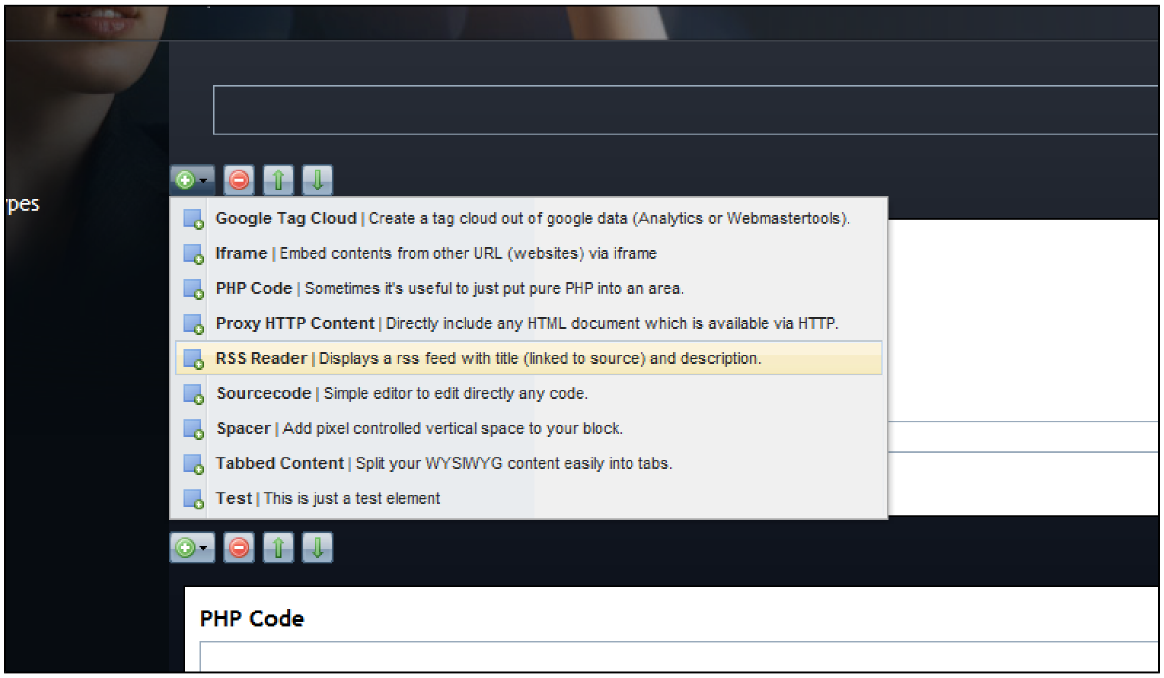
```
<?php echo $this->input('iframe_src', array(
    'grow' => true,
    'cls' => 'my-css-class'
)); ?>
```

You can also use Zend_Json_Expr to add "native" Javascript to an editable:

```
<?php echo $this->input('iframe_src', array(
 'validator' => new Zend_Json_Expr('
    function(value){
        if(value.match(/http:.*/)){
            return true;
        }else{
            return "invalid";
        }
    }')
)); ?>
```

Areablock (since 1.3.2)

The areablock is the content construction kit for documents offered by pimcore.
The concept is like you know it already from the block element. The difference is that you can insert predefined "mini applications" called bricks into an areablock.


Google Tag Cloud | Create a tag cloud out of google data (Analytics or Webmastertools).
Iframe | Embed contents from other URL (websites) via iframe
PHP Code | Sometimes it's useful to just put pure PHP into an area.
Proxy HTTP Content | Directly include any HTML document which is available via HTTP.
RSS Reader | Displays a rss feed with title (linked to source) and description.
Sourcecode | Simple editor to edit directly any code.
Spacer | Add pixel controlled vertical space to your block.
Tabbed Content | Split your WYSIWYG content easily into tabs.
Test | This is just a test element

Integrate an areablock in a template

Similar to the other document editables, an areablock can be integrated in any document view template as follows:

```
<?php echo $this->areablock('myAreablock') ?>
```

advanced usage with allowed areas:

```
<?php echo $this->areablock("myAreablock",array(
    "allowed"=>array("iframe","googletagcloud","spacer","rssreader"),
    "group" => array(
        "First Group" => array("iframe", "spacer"),
        "Second Group" => array("rssreader")
    ),
    "areablock_toolbar" => array(
        "title" => "",
        "width" => 230,
        "x" => 20,
        "y" => 50,
        "xAlign" => "right",
        "buttonWidth" => 218,
        "buttonMaxCharacters" => 35
    ),
    "params" => array(
        "iframe" => array( // some additional parameters / configuration for the brick type "iframe"
"parameter1" => "value1",
            "parameter2" => "value2"
        ),
        "googletagcloud" => array( // additional parameter for the brick type "googletagcloud"
"param1" => "value1"
        )
    )));
?>
```

Accessing the parameters by name from within the brick:

```
//echo the value of parameter named "param1" for this brick
echo $this->param1;
```

Configuration

Name	Type	Description
allowed	array	An array of area-ID's wich are allowed for this tag
params	array	Optional Parameter, this can also contain additional brick-specific configurations, see "brick-specific configuration"
group	array	Array with group configuration (see example above)
manual	bool	forces the manual mode, which enables a complete free implementation for areablocks, for example using real <table> elements ... example see blelow (since 1.4.5)
reload	bool	set to true, to force a reload in editmode after reordering items (default: false) (since 1.4.5)
toolbar	bool	set to false to not display the extra toolbar for areablocks (default: true) (since 1.4.5)
dontCheckEnabled	bool	set to true to display all installed area bricks, regardless if they are enabled in the extension manager
limit	int	limit the amount of elements
areablock_toolbar	array	Array with option that allowes you to change the position... of the toolbar

Brick-specific configuration (since 1.4.7)

Brick-specific configurations are passed using the *params* configuration (see above).

Name	Type	Description
forceEditInView	bool	if a brick contains an edit.php there's no editmode for the view.php, if you want to have the editmode enabled in both templates, enable this option

Example:

```
<?= $this->areablock("myArea", array(
    "params" => array(
        "my_brick" => array(
            "forceEditInView" => true
        )
    )
)); ?>
```

Methods

Name	Description

How to create "bricks" for the areablock?

Please read more here ..."

Using manual mode (since 1.4.5)

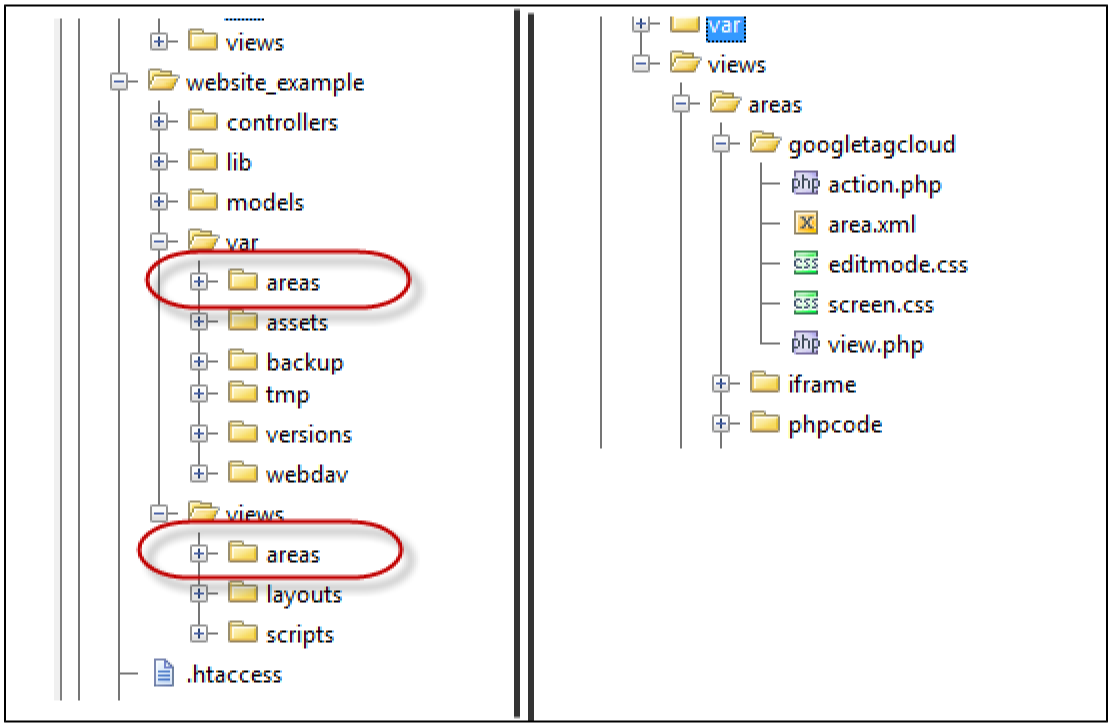
The manual mode offers you the possibility to deal with areablocks the way you like, this is for example useful with tables:

```
<?php $areaBlock = $this->areablock("myArea", array("manual" => true))->start(); ?>
<table width="100%">
    <?php while ($areaBlock->loop()) { ?>
        <?php $areaBlock->blockConstruct(); ?>
            <tr>
                <td>
                    <?php $areaBlock->blockStart(); ?>
                    <?php $areaBlock->content(); ?>
                    <?php $areaBlock->blockEnd(); ?>
                </td>
            </tr>
        <?php $areaBlock->blockDestruct(); ?>
    <?php } ?>
</table>
<?php $areaBlock->end(); ?>
```

Create your own bricks

Architecture of a brick

The architecture is simple and straightforward:



You can put your own bricks into **/website/views/areas** downloaded bricks from the marketplace are located in **/website/var/areas**

On the right side you can see how a brick can be structed. **Mandatory files are <u>area.xml</u> and <u>view.php</u>** . Optional files are <u>action.php</u>, <u>edit.php</u> and <u>icon.png.</u>

If you put an icon.png (16x16 pixel) into the brick's folder, this icon is added automatically to the toolbar, there's no need to specify the icon in the area.xml again.

The area.xml contains some meta-infos concerning the brick, and the view.php is a simple Zend_View - script where you can use all pimcore editables.

How to create a brick

First of all create the **area.xml** containing the meta-data- For example:

```
<?xml version="1.0"?>
<zend-config xmlns:zf="http://framework.zend.com/xml/zend-config-xml/1.0/">
    <id>iframe</id>
    <name>Iframe</name>
    <description>Embed contents from other URL (websites) via iframe</description>
    <!-- the icon is optional, see above for details (this node is normally used to refer to an icon
in the pimcore core icon-set) -->
    <icon>/pimcore/static/img/icon/html.png</icon>
    <version>1.0</version>
    <myCustomConfig>MyValue</myCustomConfig>
</zend-config>
```

The bold definitions are mandatory, but you can add your custom configuration for the brick as well. In the following example you can see how to access the configuration and your custom properties in the configuration (since 1.4.2).

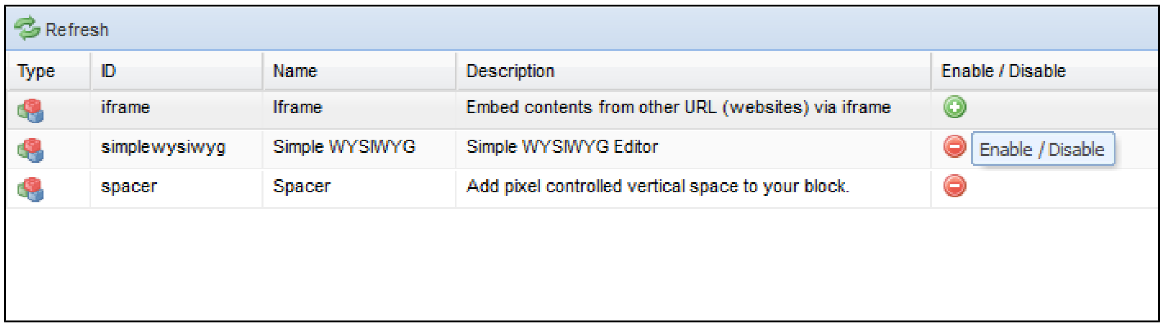
Then you have to create your view script called **view.php .**

Since pimcore 1.4.2 there is an info-objects which contains informations about the current brick. You can access this info-object in your view with **$this->brick** , the object contains the configuration from above (Zend_Config) an some other metadata (described later).

For example:

```
<?php if ($this->editmode) { ?>
    <!-- with <?php echo $this->brick->getPath(); ?> you get the path to the area out of the
info-object -->
    <link rel="stylesheet" type="text/css" href="<?php echo $this->brick->getPath(); ?>/editmode.css"
/>
    <div>
        <h2>IFrame</h2>
        <div>
            URL: <?php echo $this->input("iframe_url"); ?>
        </div>
        <br />
        <b>Advanced Configuration</b>
        <div>
            Width: <?php echo $this->numeric("iframe_width"); ?>px (default: 100%)
        </div>
        <div>
            Height: <?php echo $this->numeric("iframe_height"); ?>px (default: 400px)
        </div>
        <div>
            Transparent: <?php echo $this->checkbox("iframe_transparent"); ?> (default: false)
        </div>
    </div>
<?php } else { ?>
    <?php if (!$this->input("iframe_url")->isEmpty()) { ?>
        <?php
            // defaults
            $transparent = "false";
            $width = "100%";
            $height = "400";

            if(!$this->numeric("iframe_width")->isEmpty()) {
                $width = (string) $this->numeric("iframe_width");
            }
            if(!$this->numeric("iframe_height")->isEmpty()) {
                $height = (string) $this->numeric("iframe_height");
            }
            if($this->checkbox("iframe_transparent")->isChecked()) {
                $transparent = "true";
            }
        ?>
        <iframe src="<?php echo $this->input("iframe_url"); ?>" width="<?php echo $width; ?>" height=
"<?php echo $height; ?>" allowtransparency="<?php echo $transparent; ?>" frameborder="0" ></iframe>
    <?php } ?>
<?php } ?>
```

Once the code is in place, the areablock will appear as an extension in the Extension Manager (Extras->Extensions->Manage Extensions). From there, **you have to enable** the areablock.



Type	ID	Name	Description	Enable / Disable
	iframe	Iframe	Embed contents from other URL (websites) via iframe	⊕
	simplewysiwyg	Simple WYSIWYG	Simple WYSIWYG Editor	⊖
	spacer	Spacer	Add pixel controlled vertical space to your block.	⊖

That's all. In this example we need no action.php because everything is managed with editables.

The info-object ($this->brick) since 1.4.2

As mentioned already above, the info-object contains useful informations about the current brick.

The info-object ist available as a common view variable (***$this->brick)*** in your view scripts (edit.php and view.php).

In the follwing table you can see all available methods in your views (both in edit.php and view.php):

Method	Description
$this->brick->getId()	returns the id of the current brick
$this->brick->getConfig()	returns the configuration (Zend_Config) out of area.xml (to get your custom properties, ...)
$this->brick->getIndex()	returns the current index inside the areablock
$this->brick->getPath()	returns the (web-)path to the current brick, this is useful for embedding external stylesheets, javascripts, ...

For an example usage see the above example (referencing the stylesheet).

Configuration in Editmode (edit.php)

To allow users to add data to the brick you have to the file edit.php which can include HTML and editables. When this file is present an icon will appear for the user which can be clicked to display and edit the editable fields.

Warning: Using edit.php will **disable** all editables in view.php (they appear like in the frontend, but cannot be edited). You cannot have editables in both files. (this behavior is present in Pimcore 1.4.1)

Contents of edit.php:

```
Class: <?php echo $this->input('class'); ?>
```

Accessing the data in view.php

```
<?php
    $class = '';
    if(!$this->input('class')->isEmpty()) {
        $class = $this->input('class')->getData();
    }
?>
```

Actions for Bricks (action.php)

Sometimes a brick is more than just a view-script an contains some functionality which shouldn't be directly in the view. In this case you can use the action.php.

The action.php doesn't contain a "real" ZF - compatible controller/action it is just a little helper to get some logic and code out of the view, but the behavior is like in a common ZF-controller.

To use this feature simply create a new file called action.php in your brick directory, and insert the following code (and replace "MyBrickName" with the name of your brick):

```
<?php
class Document_Tag_Area_MyBrickName extends Document_Tag_Area_Abstract {

    public function action () {
        $myVar = $this->_getParam("myParam");
        //...
$this->view->myVar = $myVar;
    }

    /**
     * Executed after a brick is rendered
     */
    public function postRenderAction(){
        //...
}


    /**
     * Returns a custom html wrapper element (return an empty string if you don't want a wrapper
element)
     */
    public function getBrickHtmlTagOpen($brick){
        return '<span class="customWrapperDiv">';
    }

    public function getBrickHtmlTagClose($brick){
        return '</span>';
    }
}
```

The method action(); is called automatically before rendering the view.php or edit.php. Of course you can create your own methods in the class, but ensure that your class extends Document_Tag_Area_Abstract ;-)
The method postRenderAction() ist called after a brick is rendered.
The methods "getBrickHtmlTagOpen" and "getBrickHtmlTagClose" allowes you to use custom Brick wrappers (if the methods aren't specified - the regular "Pimcore-HTML-Brick-Wrappers" will be inserted)

You can access the action.php Object in your views with

```
$this->actionObject
```

Inside this class/object there are some general methods available (inherited from Document_Tag_Area_Abstract) which offers you some handy features like the info-object (as described above), the config and much more... to see it in detail check out the contents of Document_Tag_Area_Abstract, the methods are really self-describing .

For a detailed example please have a look into our examples below.

Enabling bricks

Self created bricks are extensions just like those downloaded from the repository. Before they can be used, they need to be enabled in the extension manager (Extras > Manage Extensions)

Sharing bricks with the extension-manager

If you want to share your brick using the pimcore extension manager please ensure that the brick contains all necessary javascripts and stylesheets. They should be references directly in the view (HTML), so you can use the brick "out-of-the-box" after you have installed it via the extension downloader.

Examples

You can download some examples of bricks here.

They also contain a full featured example with an action and so on (googletagcloud).

Area (since 1.4.3)

The area editable is similar to the areablock editable, the only difference is that the area bricks are not wrapped into a block element, and the editor cannot choose which area is used, this has to be done at the editable configuration in the template.

This editable is especially to use bricks also outside an areablock, for example in common block elements or just the element itself.

Configuration

Name	Type	Description
type	string	ID of the brick which should be used in this area
params	array	Optional Parameter see areablock for details

Methods

Name	Description

Example

```
<div style="margin: 20px;">
<?= $this->area("myArea", array("type" => "googleanalyticstagcloud")); ?>
</div>
```

Block

A block element is a iterating component which is really powerful.
Basically a block is only a loop, but you can use other editables in this loop, so it's possible to repeat a set of editables to create a structured page.
The items in the loop as well as their order can be defined by the editor with the block controls.

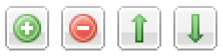
Configuration

Name	Type	Description
limit	integer	Max. amount if iterations
default	integer	If block is empty, this specifies the iterations at startup
manual	bool	forces the manual mode, which enables a complete free implementation for blocks, for example using read <table> elements ... example see blelow (since 1.4.5)

Methods

Name	Description
getCount()	Get the total amount of iterations
getCurrent()	Get the current index while looping

The Block Controls



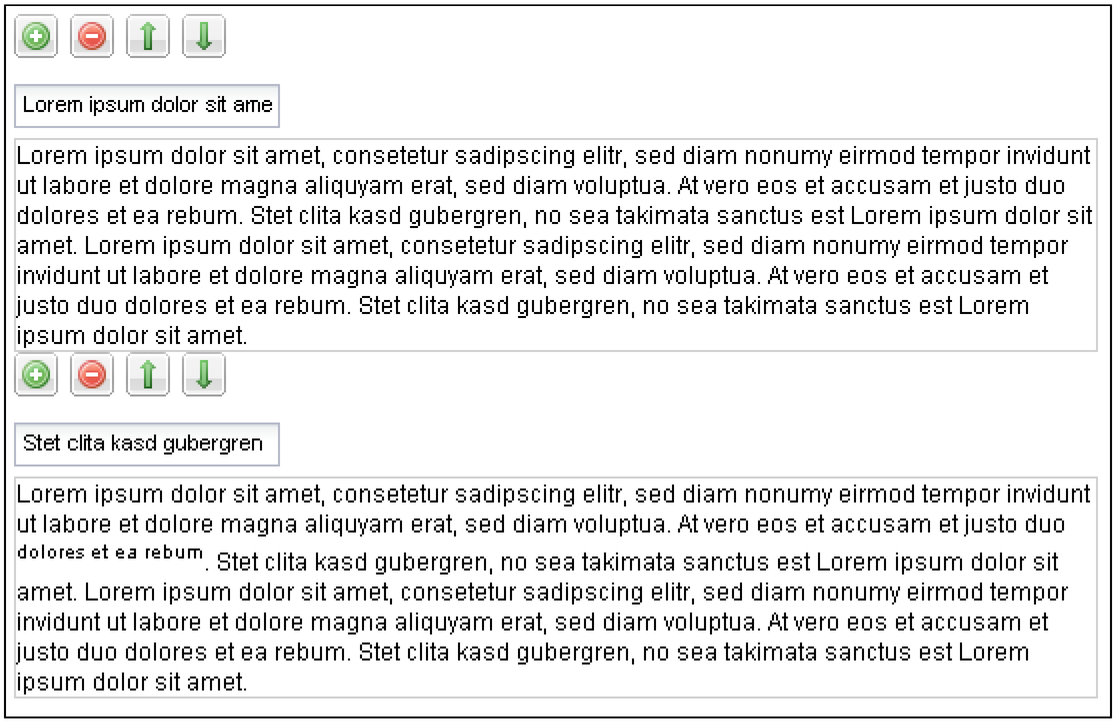
Control	Operation

⊕	Add a new block at the current position
⊖	Remove the current block
↑	Move block up
↓	Move block down

Basic usage

```
<?php while($this->block("contentblock")->loop()) { ?>
    <h2><?php echo $this->input("subline"); ?></h2>
    <?php echo $this->wysiwyg("content"); ?>
<?php } ?>
```

This will result in editmode in this:



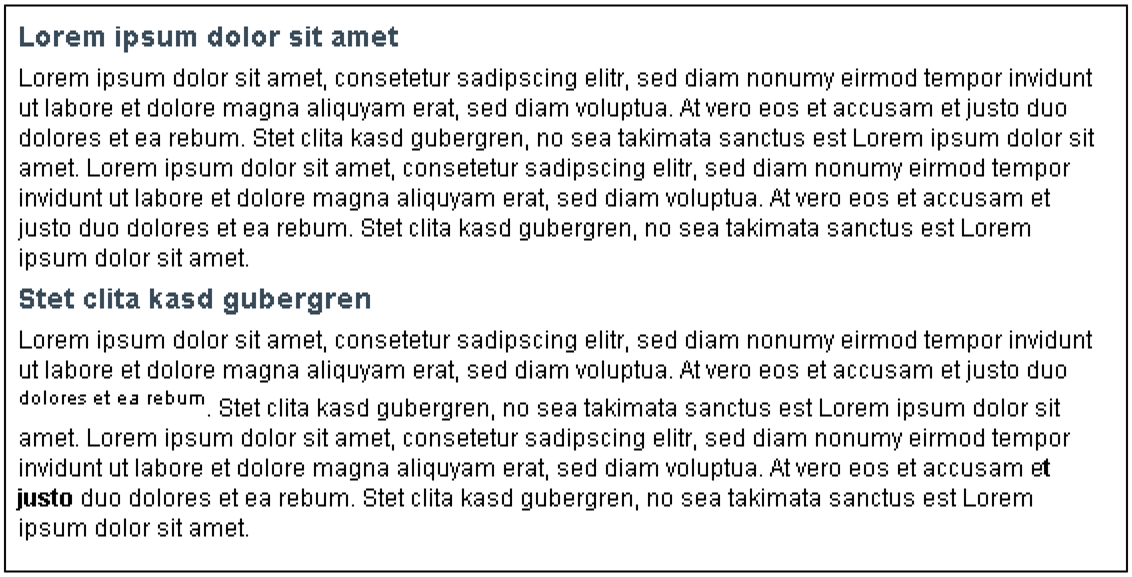
Lorem ipsum dolor sit ame

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

Stet clita kasd gubergren

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

And in the frontend:


Lorem ipsum dolor sit amet

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

Stet clita kasd gubergren

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et **justo** duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

Advanced Usage

```
<?php while($this->block("contentblock")->loop()) { ?>
    <?php if($this->editmode) { ?>
        <?php echo $this->select("blocktype",array(
            "store" => array(
                array("wysiwyg", "WYSIWYG"),
                array("contentimages", "WYSIWYG with images"),
                array("video", "Video")
            ),
            "reload" => true
        )); ?>
    <?php } ?>

    <?php if(!$this->select("blocktype")->isEmpty()) {
        $this->template("content/blocks/".$this->select("blocktype")->getData().".php");
    } ?>
<?php } ?>

<?php while($this->block("teasers",array("limit" => 2))->loop()) { ?>
    <?php echo $this->snippet("teaser") ?>
<?php } ?>
```

Example for getCurrent()

```
<?php while ($this->block("myBlock")->loop()) { ?>
    <?php if ($this->block("myBlock")->getCurrent() > 0) { ?>
        Insert this line only after the first iteration<br />
        <br />
    <?php } ?>
    <h2><?php echo $this->input("subline"); ?></h2>

<?php } ?>
```

Using manual mode (since 1.4.5)

The manual mode offers you the possibility to deal with block the way you like, this is for example useful with tables:

```
<?php $block = $this->block("gridblock", array("manual" => true))->start(); ?>
<table>
    <tr>
        <?php while ($block->loop()) { ?>
            <?php $block->blockConstruct(); ?>
                <td customAttribute="<?= $this->input("myInput")->getData() ?>">
                    <?php $block->blockStart(); ?>
                        <div style="width:200px; height:200px;border:1px solid black;">
                            <?php echo $this->input("myInput"); ?>
                        </div>
                    <?php $block->blockEnd(); ?>
                </td>
            <?php $block->blockDestruct(); ?>
        <?php } ?>
    </tr>
</table>
<?php $block->end(); ?>
```

Checkbox

Configuration

Name	Type	Description
reload	boolean	Set to true to reload the page in editmode after changing the state

Accessable Methods & Types

Name	Type	Description
value	boolean	Status of the checkbox
isChecked()	boolean	Get status of the checkbox

Simple Example

```
<?php echo $this->checkbox("myCheckbox") ?>
```

Advanced Example

```
<?php if( $this->checkbox("myCheckbox")->isChecked() ) { ?>
//Code
<? } ?>
```

Date

Basic Usage

The following code will create a simple date widget in editmode. In frontend it will output the date as defined in “output”.

Localization is automatically by registering the locale globally the ZF way or using the pimcore localization, read more here.

Simple Example

```
// simple example
<?php echo $this->date("myDate", array(
     "format" => "d.m.Y"
)); ?>
```

Configuration

Name	Type	Description
format	string	A String which describes how to output the date see [Ext.form.DateField]().

Additionally you can use every configuration property of Ext.form.DateField to customize the date widget in editmode.

Href (1 to 1 Relation)

Href provides to create a reference to an other element in pimcore (document, asset, object).
This can be useful to link a video for example (in editmode show the href to link the video out of the assets, outside embed a object code an make a reference to the video.).

In frontend-mode the href returns the path of the linked element.

Configuration

Name	Type	Description	
only	enum (document|asset|object)	Restrict to type (only one is possible)	DEPRECATED since pimcore 1.3.3
types	array	Allowed types (document, asset, object), if empty all types are allowed	Feature included since pimcore 1.3.3
subtypes	array	Allowed subtypes grouped by type (folder, page, snippet, image, video, object, ...), if empty all subtypes are allowed (see example below)	Feature included since pimcore 1.3.3
classes	array	Allowed object class names, if empty all classes are allowed	Feature included since pimcore 1.3.3
reload	true|false	true triggers page reload on each change	Feature included since pimcore 1.3.3
width	int	Width of the field in pixel	
uploadPath	string	Target path for (inline) uploaded assets (since 1.4.6)	Feature included since pimcore 1.4.6

Properties and methods

Name	Type	Description
getElement()	Document|Asset|Object_Abstract	Object assigned to the href
getFullPath()	string	Get the path of the assigned element
element	Document|Asset|Object_Abstract	The property for getElement() it's a good idea to use the getter

Examples

```
// Basic usage
<?php echo $this->href("myHref"); ?>


// Usage with restriction
<?php echo $this->href("myHref",array(
    "types"=>array("asset","object"),
    "subtypes"=>array(
        "asset" => array("video","image"),
        "object" => array("object")
     ),
    "classes" => array("myClass");
 )); ?>

// Advanced usage with getElement()
<?php if ($this->editmode) { ?>
    <?php echo $this->href("myHref"); ?>
<?php } else { ?>
    <?php
        $myHref = $this->href("myHref")->getElement();
        echo $myHref->getName();
    ?>
<?php } ?>

// Advanced usage (the video example)
<?php if ($this->editmode) { ?>
    <?php echo $this->href("myHref"); ?>
<?php } else { ?>
    <?php if ($this->href("myHref")->getElement() instanceof Asset_Video) { ?>
        <script type="text/javascript">
            var params = {
                quality: 'high'
            };
            var flashvars = {
                videoFile : "<?= $this->href("myHref")->getFullPath() ?>"
            };
            swfobject.embedSWF("/static/swf/player.swf", "videoPlayerElement", "400", "300", "9.0.0",
"", flashvars, params);
        </script>
    <?php } ?>
<?php } ?>
```

Image

Configuration

Name	Type	Description
title	string	You can give the image widget in editmode a title
width	integer	Width of the image in pixel
height	integer	Height of the image in pixel
thumbnail	string	Name of the configured thumbnail which should be used
hidetext	boolean	Hides the input for the ALT-text in editmode
reload	boolean	Set true to reload the page in editmode after updating the image
minWidth	integer	min. width of the image (in pixel)
minHeight	integer	min. height of the image (in pixel)
attributes	array	custom attributes for the <img /> tag - this can be used to pass custom attributes (not w3c) (since 1.4.6)

uploadPath	string	Target path for (inline) uploaded images (since 1.4.6)
highResolution	float	factor the thumbnail dimensions should be multiplied with (html attributes width and height contain the original dimensions ... used for "Retina" displays, print, ...)

You can also pass every valid attribute an img-tag can have (w3.org - Image), such as:

- class
- style
- ...

Methods

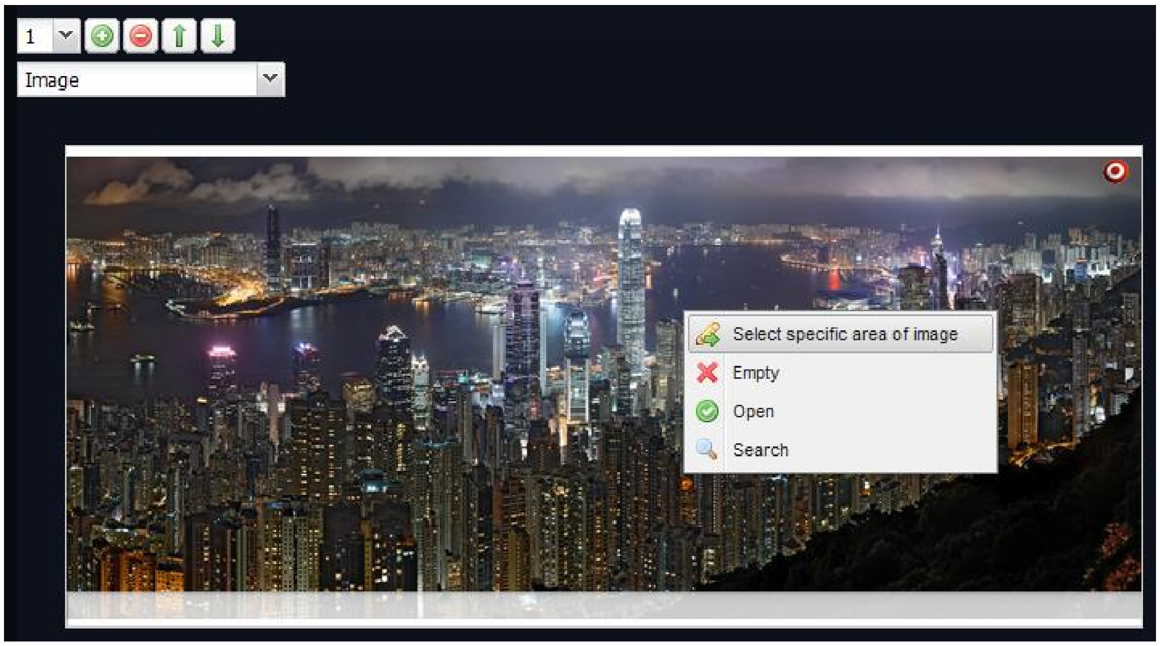
Name	Arguments	Return Value	Description
getThumbnail($name)	(string/array) $name	string (until 1.4.10), Asset_Image_Thumbnail (after 1.4.10)	get a specific thumbnail of the image
getText() / getAlt()	-	string, alt/title text from the image	The entered alternative text in the widget
getSrc()	-	string, absolute path to the image	The path to the original image which is referenced
getImage()	-	Asset_Image	The asset object which is referenced (Asset_Image) (since 1.4.6)
getHotspots()	-	array	returns the hotspot data (see example below)
getMarker()	-	array	returns the marker data (see example below)

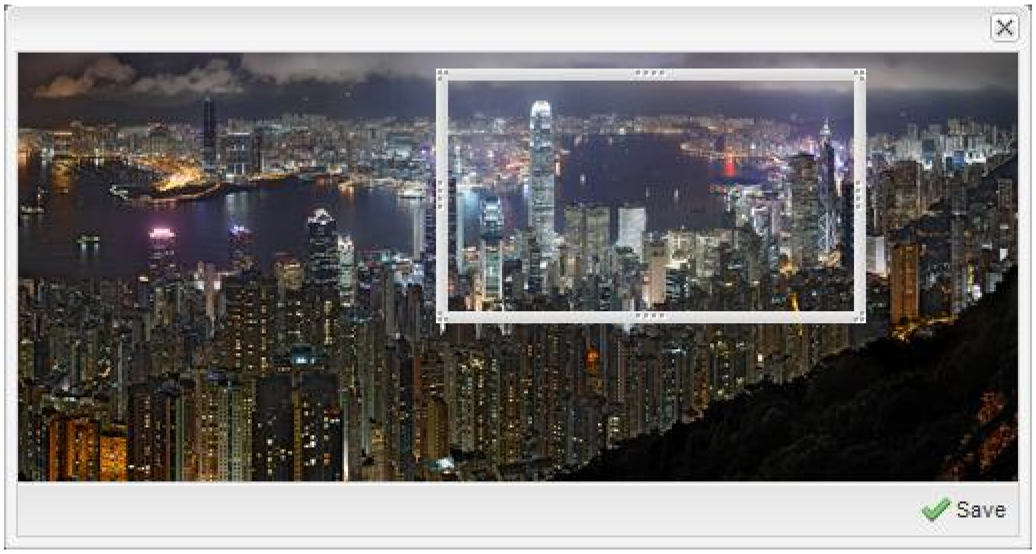
Examples

```
// Basic usage
<?php echo $this->image("myImage"); ?>

// Advanced usage
<?php echo $this->image("myImage", array(
    "title" => "Drag your image here",
    "width" => 200,
    "height" => 200,
    "thumbnail" => "contentimages"
)); ?>


// An example with a direct thumbnail configuration
<?php echo $this->image("myImage", array(
    "title" => "Drag your image here",
    "width" => 200,
    "height" => 200,
    "thumbnail" => array(
        "width" => 200,
        "height" => 200,
        "interlace" => true,
        "quality" => 90
    )
)); ?>



// example with custom attributes (since 1.4.6)
<?= $this->image("image", array(
    "thumbnail" => "contentfullimage",
    "attributes" => array(
        "custom-attr" => "value",
        "data-role" => "image"
    )
)) ?>



// get retina image (after 1.4.10)
<?php echo $this->image("myImage", array(
    "thumbnail" => array(
        "width" => 200,
        "height" => 200
    ),    "highResolution" => 2
)); ?>
// will output<img src="/website/var/thumb_9999__auto_xxxxxxxx@2x.png" width="200" height="200" />
<!-- but the real image size is 400x400 pixel -->





// custom image tag (thumbnail objects) (after 1.4.10)
<?php if($this->editmode) { ?>
    <?php echo $this->image("myImage", array("thumbnail" => "myThumbnail")); ?>
<?php } else { ?>
    <?php $thumbnail = $this->image("myImage")->getThumbnail("myThumbnail"); ?>
    <img src="<?php echo $thumbnail; ?>" width="<?php echo $thumbnail->getWidth(); ?>" height="<?php
echo $thumbnail->getHeight(); ?>" data-custom="xxxx" />
<?php } ?>
```

Fieldspecific Image Cropping for Documents (since 1.4.2)

With right click on the image in document edit mode, it is possible to define field specific image cropping.





So there is no need any more to define specific images or thumbnails if a specific region of an image should be shown. Just assign the original image and define field specific cropping directly within the document.

Marker & Hotspots (since 1.4.10)

This functionality is available on every image editable (no configuration necessary).
Setting a marker or a hotspot on an image has no direct effect on the output, the assigned image is displayed as usual.

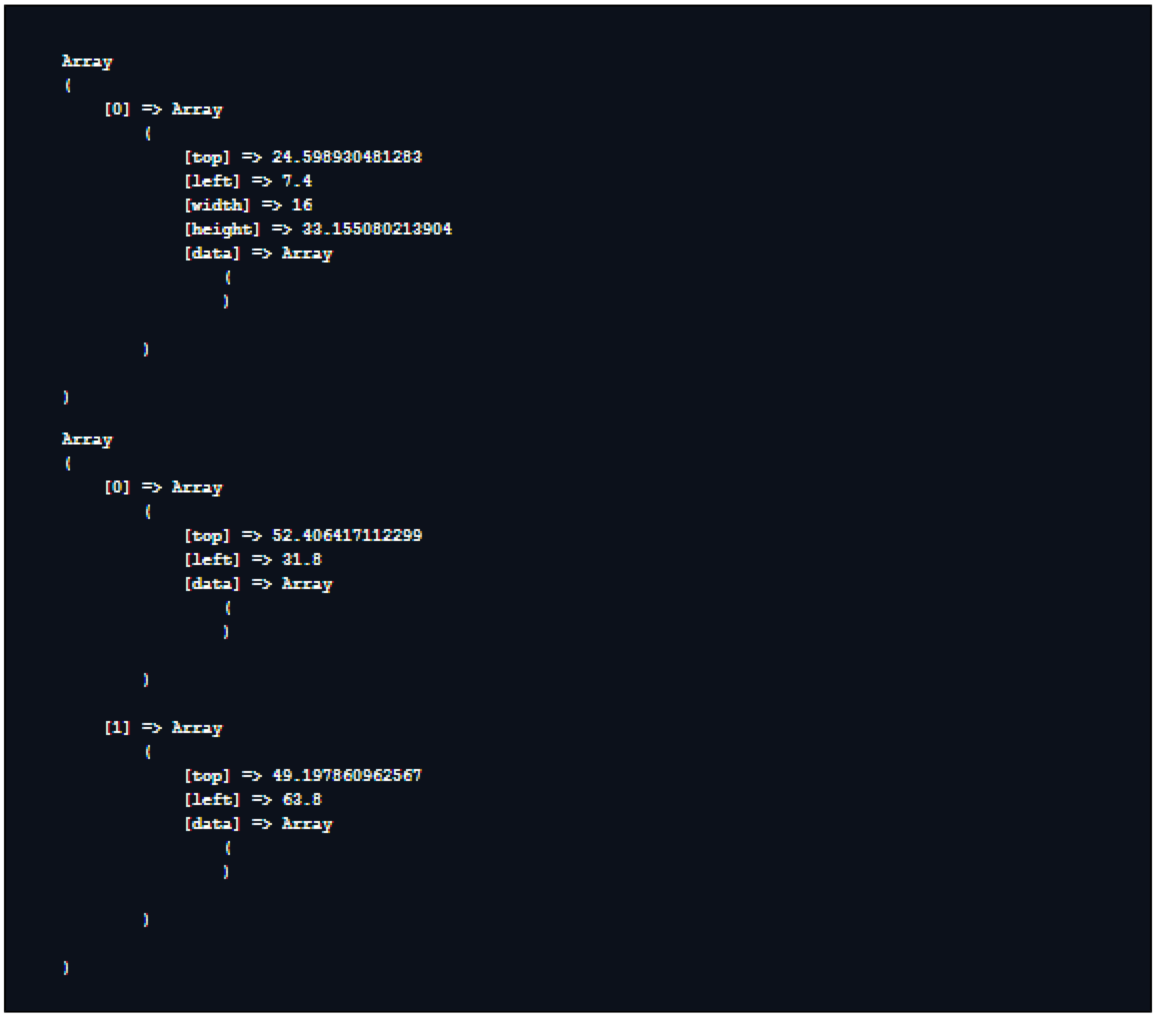
You as a developer have to get the data out of the image editable to build amazing frontends with it.

You can get the data with the methods *getMarker()* and *getHotspots().* All dimensions are in percent and therefore independent from the image size, you have to change them back to pixels according to your image size.

Example

```
<div>
 <p>
        <?= $this->image("image", array(
            "thumbnail" => "contentfullimage"
        )) ?>
        <?php if(!$this->editmode) { ?>
            <?php
                // outside the editmode: do something with the data
                if($this->image("image")->getHotspots()) {
                    p_r($this->image("image")->getHotspots());
                }
                if($this->image("image")->getMarker()) {
                    p_r($this->image("image")->getMarker());
                }
            ?>
        <?php } ?>
 </p>
</div>
```

... and here is the output:



```
Array
(
    [0] => Array
        (
            [top] => 24.598930481283
            [left] => 7.4
            [width] => 16
            [height] => 33.155080213904
            [data] => Array
                (
                )

        )

)

Array
(
    [0] => Array
        (
            [top] => 52.406417112299
            [left] => 31.8
            [data] => Array
                (
                )

        )

    [1] => Array
        (
            [top] => 49.197860962567
            [left] => 63.8
            [data] => Array
                (
                )

        )

)
```

Input

Configuration

Name	Type	Description
width	integer	Length of the input in editmode (in pixels)
htmlspecialchars	boolean	Set to false to get the raw value without HTML special chars like & (default to true)
autoStyle	bool	set to false to disable the auto-styling feature (gets the styles from the parent element and applies them to the input field) (since 1.4.6)

Additionally you can use every configuration property of Ext.form.TextField to customize the date widget in editmode.

Accessable properties

Name	Type	Description
text	string	Value of the input, this is useful to get the value even in editmode

Example

```
// Basic usage
<?php echo $this->input("headline"); ?>

//Advanced usage
<?php echo $this->input("headline", array("width" => 540)); ?>
```

Validation

It's possible to validate the input using one of the inbuilt Ext.form.VTypes ('alpha', 'alphanum', 'email' or 'url') or by making your own

```
//basic in-built validation
<?php echo $this->input("headline", array("vtype"=>"alphanum")); ?>

//advanced usage
<?php if($this->editmode): ?>
<script type="text/javascript">
    $(function(){

        // must return true or false
        Ext.apply(Ext.form.VTypes, {
          starts_with_number: function(val, field){
            if(val.match(/\d.*/)){
              return true;
            }else{
              return false;
            }
          },
          starts_with_numberText: "This field should start with a number"
        });
    });

</script>
<?php endif ?>

<?php echo $this->input("headline", array("vtype"=>"starts_with_number", "vtypeText"=>"This field is
invalid")); ?>
```

From version 1.4.2 it is possible to pass a callback into the input options using Zend_Json_Expr

```
<?php
// function must return true or an error message
echo $this->input("headline", array("width" => 540,
  "validator" => new Zend_Json_Expr('
    function(value){
      if(value.match(/\d.*/))
      {
        return true;
      }
      else
      {
        return "This field should start with a number"; }
      }')
  )); ?>
```

Link

Configuration

You can pass every valid attribute an a-tag can have (w3.org - Link), such as:

- class
- target
- id
- style
- accesskey
- name
- title
- class
- ...

Name	Type	Description
reload	boolean	Set to true to reload the page in editmode after changing the state

Methods

Name	Return-Type	Description
getHref()	string	get the path of this link
getText()	string	get the text of the link
getTarget()	string	get the target of the link
getParameters()	string	get the query params of the link
getAnchor()	string	get the anchor text of the link
getTitle()	string	get the title of the link
getRel()	string	get the rel text of the link
getTabindex()	string	get the tabindex of the link
getAccessKey()	string	get the access key of the link
isEmpty()	string	empty or not

Simple Example

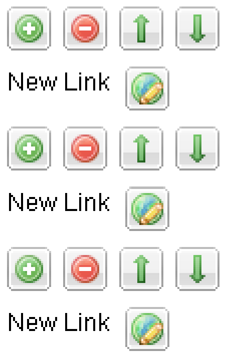
```
<?php echo $this->link("myLink"); ?>
```

Advanced Example

```
<?php while ($this->block("linkblock")->loop()) { ?>
    <?php echo $this->link("myLink", array("class" => "myClass")); ?>
<?php } ?>
```

This editable is useful for structured links, which shouldn't be inside a WYSIWYG.
It can be used with assets and documents or even as an external link starting with http://

Example linklist:



Multihref (since 1.4.2)

Multihref provides to create a references to other elements in pimcore (document, asset, object).

Configuration

Name	Type	Description
width	integer	Width for the widget in pixels (optional)
height	integer	Height for the widget in pixels (optional)
title	string	Title for the input-widget
uploadPath	string	Target path for (inline) uploaded assets (since 1.4.6)

Accessable properties

Name	Type	Description
elements	array	Array of the assigned elements

Available methods

Name	return	Description
getElements()	array	Array of the assigned elements
current()	int	Get the current index while looping

Example

```
<?php if ($this->editmode) { ?>
    <?php print $this->multihref("multihref"); ?>
<?php } else { ?>
    <!-- you can iterate through the elements using directly the tag -->
    <?php foreach($this->multihref("multihref") as $element) { ?>
        <?php echo Element_Service::getElementType($element); ?>: <?php echo $element->getFullPath();
?>
        <br />
    <?php } ?>
<?php } ?>
```

Multiselect

Configuration

Name	Type	Description
store	array	Key/Value pairs for the available options.
width	integer	
height	integer	

Additionally you can use every configuration property of Ext.ux.form.MultiSelect to customize the select widgetin editmode.

Example

```
<?php echo $this->multiselect("multiselect", array(
    "width" => 200,
    "height" => 100,
    "store" => array(
        array("value1", "Text 1"),
        array("value2", "Text 2"),
        array("value3", "Text 3"),
        array("value4", "Text 4"),
    )
)) ?>
```

Numeric

The numeric editable is like a normal textfield but with special configurations for numbers.

Configuration

Name	Type	Description
width	integer	Width of the field in pixel
minValue	float	Define a minimum value
maxValue	float	Define a maximum value

You can use every configuration property of Ext.ux.form.SpinnerField to customize the numeric widget in editmode.

Accessable Properties

Name	Type	Description
number	float	Value of the numeric field, this is useful to get the value even in editmode

Example

```
// Basic usage
<?php echo $this->numeric("myNumber"); ?>

// Advanced usage
<?php echo $this->numeric("myNumber", array(
    "width" => 300,
    "minValue" => 0,
    "maxValue" => 100,
    "decimalPrecision" => 0
)); ?>
```

Renderlet

The renderlet is a special container which is able to receive every object in Pimcore (Documents, Assets, Objects).
You can decide in your controller/action what to do with the object which is linked to the renderlet.
So it's possible to make a multifunctional dropbox in editmode where the editor can drop anything on it.

Configuration

Name	Type	Description	Mandatory
width	integer	Width of the renderlet in pixel	
height	integer	Height of the renderlet in pixel	
module	string	Specify module (default: website)	
controller	string	Specify controller	X
action	string	Specify action	X
template	string	Specify template	
title	string	Add a title to the box in editmode	
reload	bool	Reload document on change (since 1.4.2)	

Optionally you can pass every parameter (with a simple data type) you like to the renderlet which can be accessed by the configured controller with *$this->_getParam("yourKey")*.

In the configured Controller Action

In the target controller action you get the follwing parameters which can be accessed by $this->_getParam("key").

Name	Type	Description
document	Document	If the element which is dropped on the renderlet is a document this parameter is defined.
object	Object_Abstract	If the element which is dropped on the renderlet is an object this parameter is defined.
id	integer	The id of the element assigned to the renderlet
type	string	The type of the element assigned to the renderlet (document|asset|object)

subtype	string	The subtype of the element assigned to the renderlet (folder, image, link, page, classname, ...)

If you have defined custom parameters to the renderlet configuration you can access them also with $this->getParam()

Basic Example

```
<?php echo $this->renderlet("gallery", array(
    "controller" => "content",
    "action" => "gallery",
    "title" => "Drag an asset folder here to get a gallery",
    "height" => 400
)); ?>
```

Full Example

```
// code in the template / view
// this goes in the block view
<?php echo $this->renderlet("gallery", array(
    "controller" => "content",
    "action" => "gallery",
    "title" => "Drag an asset folder here to get a gallery",
    "height" => 400
)); ?>

 // and this goes in the content view in the scripts/content- folder

<?php
foreach ($this->assets as $asset) {
 echo '<img src="'.$asset.'"/>';
}
?>

// code in the controller/action

<?php

class ContentController extends Website_Controller_Action {


    public function galleryAction () {

        if($this->_getParam("type") == "asset") {
            if($asset = Asset::getById($this->_getParam("id"))) {
                if($asset->getType("folder")) {
                    $childs = $asset->getChilds();
                    $this->view->assets = $childs;
                }
            }
        }
    }
}

?>
```

Editmode awareness
Please be aware, that the renderlet itself is not editmode-aware. If you need to determine within the renderlet whether in editmode or not, you need to pass that parameter to the renderlet.

Eg:

```
$this->renderlet("myRenderlet", array(
....
'editmode' => $this->editmode
));
```

Within the renderlet, you can access the editmode parameter as follows:

```
$this->_getParam("editmode")
```

Select

Configuration

Name	Type	Description
store	array	Key/Value pairs for the available options.
reload	bool	Set true to reload the page in editmode after selecting an item

Additionally you can use every configuration property of Ext.form.ComboBox to customize the select widget in editmode.

Accessable Properties

Name	Type	Description
text	string	Value of the select, this is useful to get the value even in editmode

Example

```
// To preselect "option2" in editmode
<?
if($this->editmode){
    if($this->select("mySelect")->isEmpty()){
 $this->select("mySelect")->setDataFromResource("option2");
    }
}
?>

// Basic usage
<?php echo $this->select("mySelect",array(
    "store" => array(
        array("option1", "Option One"),
        array("option2", "Option Two"),
        array("option3", "Option Three")
    )
)); ?>

// Advanced usage
<?php echo $this->select("blocktype",array(
        "store" => array(
        array("wysiwyg", "WYSIWYG"),
        array("contentimages", "WYSIWYG with images"),
        array("video", "Video")
    ),
    "reload" => true
)); ?>
```

Snippet (embed)

Use the snippet editable to embed a document snippet, for example teasers, boxes, footers, etc.

Snippets are like little documents which can be embedded in other documents. You have to create them the same way as other documents.

It is possible for users to drag snippets onto a document (like a sidebar item that is different on every page) or for the developer to place one on a fixed position in a (layout) template (like footer that is the same on every page, see code example).

Creation

To create your own snippet start with creating a PHP file in the directory website/views/scripts/snippets. This file can contain HTML and PHP code. You can add text input fields here. This file is the view that is being used for this snippet.

The file website/controllers/SnippetsController.php is the controller and contains the actions associated with all snippets. If you have named your view "footer.php" you should add a method "footerAction()" here. This method will be called every time the snippet is displayed. You can retrieve information from the database here and pass it on to the view here using *$this->view* and adding variables to it.

After creating the view and the action the snippet will not yet appear for the user. Before a snippet can be used you need to define it as a custom Document Type.

Configuration

Name	Type	Description
width	integer	Width of the snippet in pixel
height	integer	Height of the snippet in pixel
title	string	You can give the element a title
defaultHeight	integer	A default height if the element is empty
reload	bool	Reload document on change (since 1.4.2)

Accessible Properties

Name	Type	Description
id	integer	ID of the referenced Document_Snippet
snippet	Document_Snippet	ID of the referenced Document_Snippet

Example

```
// Define a place for a snippet to be dragged onto, basic usage
<?php echo $this->snippet("mySnippet") ?>

// Define a place for a snippet to be dragged onto, advanced usage
<?php echo $this->snippet("mySnippet", array("width" => 250, "height" => 100)) ?>
```

Display text from snippet on every page

If you have a footer text that you want to appear on every page, like for example a copyright notice that the user should be able to edit, create a snippet as follows:

```
<?php echo $this->wysiwyg('footer', array('width' => 500, 'height' => 100)); ?>
```

Place the file in the snippets directory and name the file footer.php then add it a s a document type and create a snippet called "footer" as a document.

It is important that this snippet is never moved, renamed or deleted. To prevent this you can set the Document permissions so that certain users can't perform these actions. To remove the snippet from the website it can be unpublished by the user.

Then add the following code to your layout template (located in the "layouts" directory):

```
<?php
$s = Document_Snippet::getByPath('/snippets/footer');
if(is_object($s) && is_object($s->elements['footer'])) {
    echo $s->elements['footer']->frontend();
}
?>
```

This code loads the snippet from a specific location. It then checks if it indeed exists and continues to extract the text from the WYSIWYG-field named "footer" and echoes it on the page.

Security note: because we are using a WYSIWYG-field, HTML can be inserted by the user. If you don't want this; use another kind of input field. Don't forget to escape its contents before displaying them!

Table

Configuration

Name	Type	Description
width	integer	Width of the field in pixel
height	integer	Height of the field in pixel
defaults	array	Array can have the following properties: rows, cols, data (see example)

Accessable Methods & Properties

Name	Type	Description
getData()	array	Get the data of the table as array

Example

```
<?php echo $this->table("tableName",array(
    "width" => 700,
    "height" => 400,
    "defaults" => array(
        "cols" => 6,
        "rows" => 10,
        "data" => array(
            array("Value 1", "Value 2", "Value 3"),
            array("this", "is", "test")
        )
    )
)) ?>
```

Textarea

Configuration

Name	Type	Description
width	integer	Width of the textarea in pixel
height	integer	Height of the textarea in pixel
nl2br	boolean	Set to true to get also breaks in frontend
htmlspecialchars	boolean	Set to false to get the raw value without HTML special chars like & (default: true)
autoStyle	boolean	set to false to disable the auto-styling feature (gets the styles from the parent element and applies them to the input field) (since 1.4.6)

Additionally you can use every configuration property of Ext.form.TextArea to customize the date widget in editmode.

Accessable Properties

Name	Type	Description
text	string	Value of the textarea, this is useful to get the value even in editmode

Example

```
// Basic usage
<?php echo $this->textarea("myTextarea") ?>

// Advanced usage
<?php echo $this->textarea("myTextarea", array("width" => 300, "height" => 200)) ?>
```

Video

Configuration

Name	Type	Description
width	integer	Width of the video in pixel
height	integer	Height of the video in pixel
config	string/array	**Configuration for the flowplayer** Either as string or as array, use string if you want to refer to a local Javascript var, which contains the configuration. (more see examples)
youtube	array	Parameters for youtube integration. Possible parameters: https://developers.google.com/youtube/player_parameters (since v. 1.4.8)
swfPath	string	Define a different flowplayer path (eg. if you have a commercial license)
scriptPath	string	Define a different flowplayer embed script path.
thumbnail	string	Name of the video-thumbnail (required when using automatic-transcoding of videos) see: Video Thumbnails (since v 1.4.3)
imagethumbnail	string	Name of the image-thumbnail, this thumbnail config is used to generate the preview image (poster image), if not specified pimcore tries to get the information out of the video thumbnail. see also: Video Thumbnails (since v 1.4.8)
html5	bool	Only works in combination with the thumbnail configuration, if specified pimcore generates HTML5 markup for the video, this can then be used with HTML5-Videoplayers like Mediaelement.js, Projekktor, VideoJS, ... (not included)
disableProgressReload	bool	Parameter to disable the automatically page-reload if a thumbnail was not jet created (since v 1.4.8)

When using the HTML5 player don't forget to set the right mime-type. To do this, put the following lines into your .htaccess:

AddType video/mp4 .mp4
AddType video/webm .webm

Methods

Name	Arguments	Return Value	Description
getImageThumbnail($name)	(string/array) $name	string, absolute path to the thumbnail	get a specific image thumbnail of the video, or a thumbnail of the poster image (if assigned)
getVideoType()	-	string, type of the video (asset|youtube|vimeo|url)	this is to check which video type is assigned
getVideoAsset()	-	asset	returns the video asset object if assigned, otherwise null

getThumbnail()	(string/array) $name	array, absolute paths to the different video thumbnails	get a specific video-thumbnail of the video
			
			Return Value:
			
			`[status] => finished`
			`[formats] => Array`
			` (`
			`  [mp4] => /website/var /tmp/video_3414__example.mp4`
			`  [webm] => /website/var /tmp/video_3414__example.webm`
			` )`

getPosterAsset()		Asset	returns the assigned poster image asset

Accessable Properties

Name	Type	Description
id	string	Asset-ID, YouTube-URL, Vimeo-URL, External-URL, ...
type	string	One of asset, youtube, vimeo, url, ...

Example (default configuration)

```
<?php // with direct configuration as array ?>
<?php echo $this->video("myVideo", array(
    "width" => 670,
    "height" => 400,
    "config" => array(
        "clip" => array(
            "autoPlay" => false
        )
    )
)); ?>



<?php // with configuration as local javascaript variable ?>
<script type="text/javascript">
    var myVideoConf = {
        clip: {
            autoPlay: false,
            autoBuffering: true
        },
        key: "xxxxxxx",
        canvas: {backgroundColor: "transparent"},
        plugins: {
           controls: {
              durationColor: '#ffffff',
              tooltipTextColor: '#ffffff',
              sliderColor: '#000000',
              tooltipColor: '#5F747C',
              volumeSliderGradient: 'none',
              buttonOverColor: '#00466e',
              backgroundGradient: [0.6,0.3,0,0,0],
              progressGradient: 'none',
              buttonColor: '#00466e',
              timeColor: '#ffffff',
              timeBgColor: '#00466e',
              borderRadius: '0px',
              bufferGradient: 'none',
              backgroundColor: '#a4c1d4',
              volumeSliderColor: '#000000',
              progressColor: '#112233',
              bufferColor: '#00466e',
              sliderGradient: 'none',
              height: 24,
              opacity: 1.0
           }
        }
    };

</script>

<?php echo $this->video("myVideo", array(
    "width" => 670,
    "height" => 400,
    "config" => "myVideoConf"
)); ?>
```

Automatic Video transcoding (using default Flowplayer with iDevice Fallback)

```
<?php // using thumbnails with automatic video transcoding (requires FFMPEG) ?>
<?php echo $this->video("myVideo", array(
 "width" => 670,
 "height" => 400,
 "thumbnail" => "example"
)); ?>
```

HTML 5 Example with automatic Video transcoding (using mediaelement.js)

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>

    <?php if (!$this->editmode) { ?>
        <script type="text/javascript" src="jquery.js"></script>
        <script type="text/javascript" src="mediaelement-and-player.min.js"></script>
        <link rel="stylesheet" href="mediaelementplayer.css" type="text/css" media="screen" />
    <?php } ?>

</head>
<body>

    <?=  $this->video("myVideo", array(
        "thumbnail" => "example",
        "html5" => true,
        "width" => 400,
        "height" => 300
    )); ?>

    <?php if (!$this->editmode) { ?>
        <script type="text/javascript">
            $('video,audio').mediaelementplayer(/* Options */);
        </script>
    <?php } ?>

</body>
</html>
```

YouTube Example (since 1.4.8)

```
<div>
    <?= $this->video("myVideo", array(
        "thumbnail" => "example",
        "html5" => true,
        "youtube" => array(
            "autoplay" => 1,
            "modestbranding" => 1
        )
    )); ?>
</div>
```

WYSIWYG

Configuration

Name	Type	Description
width	integer	Width of the field in pixels

height	integer	Height of the field in pixels
toolbar	string	Available options: Full, Basic
resize_disabled	boolean	Set true to disable resizing
~~sharedtoolbar~~	~~boolean~~	~~Set to false to disable the "sticky" toolbar, and display it directly at the editor.~~ (as of 1.4.9)
enterMode	integer	Set it to 2 if you don't want to add the P-tag
contentsCss	string	Path to stylesheet with text styles
customConfig	string	Path to Javascript file with configuratin for CKEditor
inline	bool	set "false" to disable inline mode (contenteditable) - (since 1.4.9)

Accessable Properties

Name	Type	Description
text	string	Value of the WYSIWYG, this is useful to get the value even in editmode

See the 2nd example of the following code block.

Basic Example

```
// Basic usage
<?php echo $this->wysiwyg("myWysiwyg") ?>

// Advanced useage
<?php echo $this->wysiwyg("content", array(
    "width" => 700,
    "height" => 500));
?>
```

Custom Configuration of CKeditor

```
// It's possible to full customize the whole editor. All arguments which are defined in the
configuration are passed to the CKeditor config. For example:
<?php echo $this->wysiwyg("content", array(
    "width" => 700,
    "height" => 500,
    "customConfig" => "/custom/ckeditor_config.js"));
?>

// with the follwing code you can get the text even in editmode
<?php echo $this->wysiwyg("content", array(
    "width" => 700,
    "height" => 500,
    "customConfig" => "/custom/ckeditor_config.js"));
?>

// output the text also in editmode
<?php if ($this->editmode) { ?>
    <?php echo $this->wysiwyg("content")->text ?>
<?php } ?>
```

Example Custom Config (ckeditor_config.js) for CKeditor

```
CKEDITOR.editorConfig = function( config )
{

    config.colorButton_colors = "000000,ffffff,e60a0a,143ca0,565a5e";
    config.colorButton_enableMore = false;
    config.toolbar = [
                    ['Cut','Copy','Paste','PasteText','PasteFromWord','-', 'SpellChecker', 'Scayt'],
                    ['Undo','Redo','-','Find','Replace','-','SelectAll','RemoveFormat'],
                    ['ImageButton'],
                    ['Bold','Italic','Underline','Strike','-','Subscript','Superscript'],
                    '/',
                    ['BulletedList'],
                    ['JustifyLeft','JustifyCenter','JustifyRight'],
                    ['Link','Unlink','Anchor'],
                    ['Image','Table','SpecialChar'],
                    ['TextColor','BGColor'],
                    ['Maximize', 'ShowBlocks']
    ];
};
```

This example shows how to restrict available text colors and shows a custom toolbar config. More examples and config options for the toolbar and toolbarGroups can be found at http://nightly.ckeditor.com/13-02-21-09-53/basic/samples/plugins/toolbar/toolbar.html

Please refer to the CKeditor 4.0 Documentation

Helpers (Available View Methods)

$this->inc(mixed $document, [array $params])

Use *$this->inc()* to include documents inside of views, for example a snippet.
This is useful for footers, headers, navigations, sidebars, ...

Arguments

$document can be either an ID, a path or even the Document object itself
$params is optional and should be an array with key value pairs like in *$this->action()* from ZF.

```
<!-- include path -->
<?= $this->inc("/shared/boxes/buttons") ?>

<!-- include ID -->
<?= $this->inc(256) ?>

<!-- include object -->
<?php

$doc = Document::getById(477);
echo $this->inc($doc, array(
    "param1" => "value1"
));

?>
```

$this->template(string $path, [array $params])

This method is designed to include an other template directly, without calling an action.

```
<?php $this->template("includes/footer.php") ?>

<!-- with parameters -->
<?php $this->template("includes/somthingelse.php", array(
    "param1" => "value1"
)) ?>
```

$this->getParam(string $key)

Get's a parameter (get, post, ...), it's an equivalent to $this->_getParam() in the controller action.

```
<?php if ($this->param1) { ?>
    Some Output
<?php } ?>
```

$this->cache(string $key, [int $lifetime])

This is an implementation of a direct in-template cache. You can use this to cache some parts directly in the template, independend of the other global defineable caching functionalities. This can be useful for templates which need a lot of calculation or require a huge amount of objects. Lifetime is expected in seconds. If you define no lifetime the behavior is like the outputcache, so if you make any change in pimcore, the cache will be flushed. When specifing a lifetime this is independend from changes in the CMS.

```
<?php if(!$this->cache("test_cache_key", 60)->start()) { ?>
    <h1><?= $this->input("headline", array("width" => 670)); ?></h1>
    <?= microtime() ?>
    <?php $this->cache("test_cache_key")->end() ?>
<?php } ?>
```

$this->glossary()

Please see the glossary documentation.