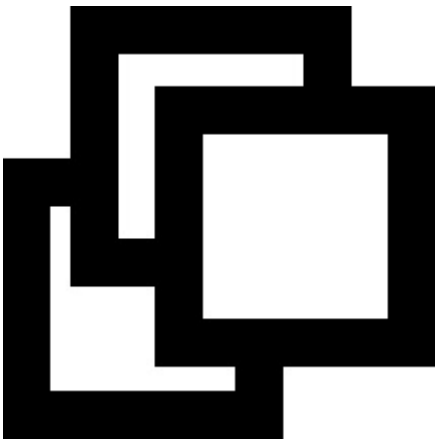
CHG instruments

BlogManager 1.0.134

doc. version: 1.134.425 - App version: 1.0.134 - Markparser version: 1.0.4.25



QUICK START

1. Menu File > Copy templates... select a folder where templates will be copied into.
2. Menu File > New site... choose the same folder where you put templates. **Site Settings** pops up: fill the required fields: Site name and URL, optional fields (description, keywords, default author name, etc.), finally check the pre-filled fields under Generation: where to save (generate) site and documents, where common images, templates, web fonts are.
3. Click Add new enter name of the document, and select Article or Page.
   Articles are indexed and ordered, Pages are unrelated.
4. Enter text and contents to your new page using the edit box on the right.
5. Click Edit entry page enter text and contents using the edit box on the right.
   This is the Welcome Content that is shown at the first index.html page (if more index#.html pages are made, welcome content is shown in the first only).
6. Hide/unhide documents to keep them private.

Save your page: CTRL+S
Save your site: CTRL+W
Show special characters: F12
Toggle error checking: CTRL+F4
Preview current page: F5
Switch pane: F8

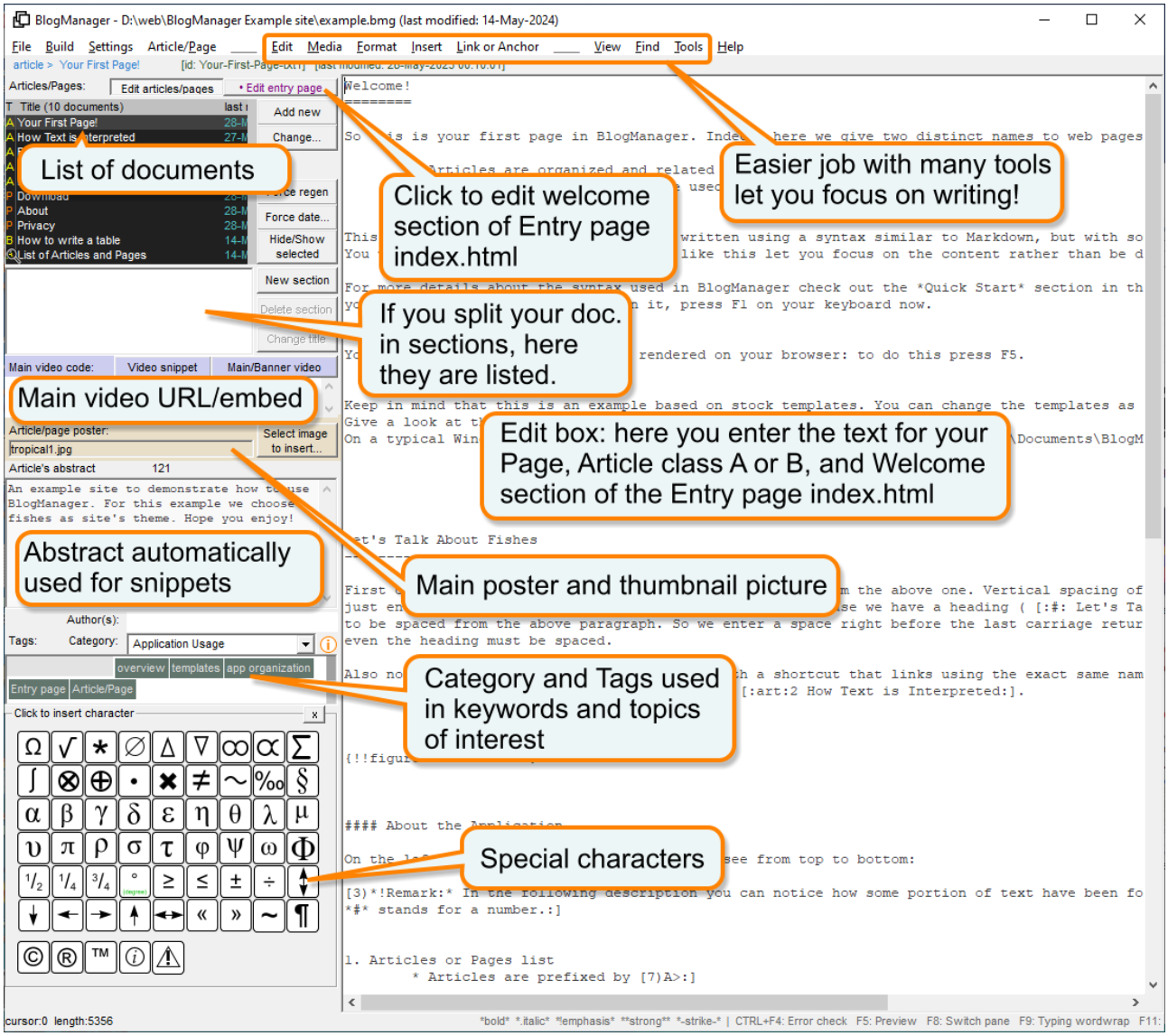
Insert Link: CTRL+L
Insert media (video): CTRL+M
Edit video metadata: CTRL+ALT+M

Classic edit controls:
CTRL+C copy; CTRL+X cut; CTRL+V paste; CTRL+Z undo/redo; CTRL+A select all
Toggle auto word-wrapping while typing: F9
Word-wrap selected paragraph: F11

Publish website: Menu Generate > Generate All.
Use Delete and generate all, to refresh everything. Use Generate incrementally to keep older versions.

Review and modify templates to fit your taste and needs.

Application Overview



GETTING STARTED - Text Examples

Text using markers syntax	Corresponding HTML	As it is rendered in browsers
`Heading` `=======` `Sub-heading` `----------` `# Alternative heading` `## Alternative sub-heading` `###### Small heading` `Paragraphs are separated` `by a blank line.` `   Tabs before paragraph make` `   it indented.` `   Space+underscore _` `   force line continuation.` `   This: -.- Force line break.` `      More tabs indent the` `   single line.` `. : ? !` at the end of the line cause line break. Append a space (and an optional underscore) to let the line to continue.	`<h2 id="Heading">Heading</h2>` `<h3 id="Sub-heading">Sub-heading</h3>` `<h1>Alternative heading</h1>` `<h2>Alternative sub-heading</h2>` `<h6>Small heading</h6>` `<p>Paragraphs are separated by a blank line.</p>` `<p style="padding-left:8ch">Tabs before paragraph make it indented. ` `Space+underscore force line continuation. ` `This:  ` `Force line break. ` `<span style="padding-left:8ch"> </span>More tabs indent the single line.</p>`	Heading Sub-heading **Alternative heading** Alternative sub-heading **Small heading** Paragraphs are separated by a blank line. Tabs before paragraph make it indented. Space+underscore force line continuation. This: Force line break. More tabs indent the single line.
`Text attributes: *!emphasis*,` `**important**, *-deleted-*,` `*.italic*, *bold*.` `Horizontal rule:` `---` `[:c{` `This paragraph is code.` `}--]` `Escape character: \*not bold*.`	`<p>Text attributes:` `<em>emphasis</em>,` `<strong>important</strong>,` `<s>deleted</s>,` `<i>italic</i>, <b>bold</b>.</p>` `<p>Horizontal rule:</p>` `<hr>` `<code>This paragraph is code.</code>` `<p>Escape character: *not bold*.</p>`	Text attributes: *emphasis*, ***important***, ~~deleted~~, *italic*, **bold**. Horizontal rule: --- `This paragraph is code.` Escape character: *not bold*.
Note on text attributes: All attributes are rendered following the style set in bm_styles.css file. Escape character can be used to escape (ignoring) every marker so it is rendered as is.		
`Bullet lists nested within` `numbered list. Lists must be` `ended with almost a blank line,` `but not nested lists:` `1. fruits` `     * apple` `     * banana` `2. meals` `     * beef` `     * lasagna` `3. tableware` `4. cookware`	`<p>Bullet lists nested within` `numbered list: Lists must be` `ended with almost a blank line,` `including nested lists:</p>` `<ol><li>fruits<ul>` `  <li>apple</li>` `  <li>banana</li></ul></li>` `  <li>meals<ul>` `  <li>beef</li>` `  <li>lasagna</li></ul></li>` `  <li>tableware</li>` `  <li>cookware</li>` `</ol>`	Bullet lists nested within numbered list. Lists must be ended with almost a blank line, including nested lists: 1. fruits • apple • banana 2. meals • beef • lasagna 3. tableware 4. cookware
Note: Numbers of ordered lists must be at the beginning of the line, with no tabulation or spaces.		

Notes: Entering more blank lines automatically create proportional line space text using embedded style.

Underline heading (using === or ---) are demoted to h3 and h4 respectively if within into a section (see Sections).

Text using markers syntax	Corresponding HTML	As it is rendered in browsers
Automatic links[1] and anchors[2] to notes. Notes: [1]: Links points to anchors. [2]: Anchors are landing points in navigation.	Automatic links [1] and anchors[2] to notes. Notes: [1]: Links points to anchors. [2]: Anchors are landing points in navigation.	Automatic links[1] and anchors[2] to notes. Notes: [1]: Links points to anchors. [2]: Anchors are landing points in navigation.
http://autolink.tld [:@:http://URL.tld (title) text:] [:@2 Link to article/page 2:] See note[1]. [1]: It's easier to insert links and anchors using the in-app tools. Anchors are automatically generated for headings.	http://autolink.tld text Link to article/page 2 See note [1]. [1]: It's easier to insert links and anchors using the in-app tools. Anchors are automatically generated for headings.	http://autolink.tld text Link to article/page 2 See note[1]. [1]: It's easier to insert links and anchors using the in-app tools. Anchors are automatically generated for headings. ***Also see***: Inserting Links
This links to [:#: Shortcuts :]. Shortcuts --------- Shortcuts are simpler links to [:#:anchors:] and headings. [:#anchors:] This is an example of an anchor.	This links to Shortcuts. <h3 id="Shortcuts">Shortcuts</h3> Shortcuts are simpler links to anchors This is an example of an anchor.	This links to Shortcuts. Shortcuts Shortcuts are simpler links to anchors and headings. This is an example of an anchor.

Text using markers syntax	Corresponding HTML	As it is rendered in browsers
Apply styles: select text and choose desired style from menu. Enumerated styles can be customized in bm_styles.css. [0)This is styled with style zero.:] [6)This is styled with style six.:]	Apply styles: select text and choose desired style from menu. Enumerated styles can be customized in bm_styles.css. This is styled with style zero. This is styled with style six.	Apply styles: select text and choose desired style from menu. Enumerated styles can be customized in bm_styles.css. This is styled with style zero. This is styled with style six.
*>Simple quotation *>(First sentence in http://www.george-orwell.org/1984/0.html "Nineteen Eighty-Four" by George Orwell \(Part 1, Chapter 1\).) It was a bright cold day in April, and the clocks were striking thirteen. *>("Albert Einstein.")God does not play dice.	<blockquote> Simple quotation </blockquote> <figure class="bm_quot"><blockquote> It was a bright cold day in April, and the clocks were striking thirteen. </blockquote><figcaption> First sentence in <cite>Nineteen Eighty-Four</cite> by George Orwell (Part 1, Chapter 1). </figcaption></figure> <figure class="bm_quot"><blockquote> God does not play dice. </blockquote><figcaption> <cite>Albert Einstein.</cite> </figcaption></figure>	Simple quotation «It was a bright cold day in April, and the clocks were striking thirteen.» First sentence in *Nineteen Eighty-Four* by George Orwell (Part 1, Chapter 1). «God does not play dice.» *Albert Einstein.*
Tabbed text in a single paragraph. Tabled text. Second column, More text still same column	Tabbed text in a single paragraph. Tabled text. More text Second column, still same column	Tabbed text in a single paragraph. Tabled text. Second column, More text still same column

Notes: For a complete reference of all recognized markers see Markers. Shortcut links: read more in Inserting Links.
See also: Inserting Pictures, Inserting Videos, Implicit tables, Explicit tables, Links and anchors.

In addition it is possible to insert mathematic formulae and chemical expressions using a similar syntax of x-Tex or machem. For more see Math Equations and Chemical Formulae.

What is BlogManager?

With BlogManager you can create static websites that do not rely on server side scripts and applications, apart for limited functions such as registering users, receiving forms or sending mail forms from server. Sites are faster and more robust.

BlogManager keeps your site organized and keeps track of several metadata such as the last modification date, article's author, and more, without encumbering your source text documents with this information.

How it works?

Just write your articles or pages with plain text, with no frills that cause distractions. It helps you stay focused on the content rather than on its aspect. BlogManager interprets your text converting it into proper and technically correct HTML. The content and metadata are then built using templates that you can customize to suit your taste and needs.

Example of text (on the left) and how it is rendered in a HTML page (on the right):

```
How to write an article or page
===============================

You can write your content in plain
text, as in this example. This will
create a piece of text with an *!
anchored* heading (a heading that can
be linked), and you can bring
attention to part of text adding:
*bold*, *.italic*, *!emphasized* and
**important** attributes to words and
phrases.

Links can be automatically detected
such as:
https://accidentalscience.com. You can
enter pictures and videos as well.
```

How to write an article or page

You can write your content in plain text, as in this example. This will create a piece of text with an *anchored* heading (a heading that can be linked), and you can bring attention to part of text adding: **bold**, *italic*, *emphasized* and ***important*** attributes to words and phrases.

Links can be automatically detected such as: https://accidentalscience.com. You can enter pictures and videos as well.

BlogManager keeps the chronology of the documents, collects references to easily link articles and pages, generates a searchable index, collect and automatically includes keywords and categories, hinting you with keywords and categories already recorded.

Finally it generates the site in a folder that mirrors what should go online. It doesn't provide a FTP client though, so you need an external application to upload your content. However it can generate a batch (.bat) script to control cURL. If you have Windows 10 or 11 cURL is already installed on your computer.

Common and Known Issues

Build modified since last generation fails

Symptom: After building the generated site lacks last added articles and their related content (pictures, videos, etc.)

Solution: When you force creation and last modified date this may cause the article and its content to fall before the date of last build, so the program think that those articles were already built and skip them. To solve this problem: *a)* let the `last modified date` be later than the last build date (visible after the file name on the top title bar) or *b)* choose `build all`.

=> On version 1.0.129 and above you can also force regeneration (and upload) for selected articles, regardless of date.

User Manual

Introduction

BlogManager allows you to organize your static website generating the required pages from templates. With it you can write your articles and pages in a clean way without bothering for the formatting details using a simple subset of **markers** similar to *Markdown*. This allows you to focus on the content you make with no distractions.

The following is an example of text:

```
Welcome to BlogManager
======================

### Summary

BlogManager allows to organize your static website generating the required pages
from templates. With it you can write your articles in a clean way without
bothering for the formatting details using a simple subset of *markers* similar to
**Markdown**.
This allows you to focus on the content you make with no distractions.
```

The written text is then automatically converted into proper and syntactically correct HTML5 elements. The generated text is then incorporated into a page template that can be customized to suit your needs. Furthermore, multiple index pages are generated to create a series of pages that summarize and link to the articles, populating an index template.
For more information see Template Files .

Within an article can be inserted images, tables, videos, and special elements using escape sequences. You can also add comments that will not be rendered in the output HTML file.

You can add an abstract for each article, and you can define a category and a list of tags (keywords) associated with each article, useful for SEO purposes.

A search page and its related files are also generated, allowing the users to search into your website by category or by tag (keyword). Full client side.

Some special markers allow to create full compliant HTML5 articles, making *header*, *sections*, *footer* entities, and *aside* entities.

Articles and Pages

In BlogManager you can create and edit *articles* and *pages*. The difference between the two is that *articles* are aggregated and they are listed in featured, related and recent blocks in the index.hml, furthermore they have an abstract, tags and categories. Articles are displayed using the *templatepost.html* and *templateArtB.html* templates.
Pages are aggregated differently, they are not listed in featured, related and recent blocks, and lack the abstract. Though, they can still have tags and a category.

Article class A and B

Furthermore with version 1.0.131 and above two classes of articles are possible: article A and article B. Structurally both are identical, however articles B preset the custom template file *templateArtB.html* .
Articles A and B are listed separatedly and two different selectors (`{%xtract:articleA%}` and `{%xtract:articleB%}`) allow to list them into different regions into the HTML page.
In addition articles B cannot be listed in recent and related fields, if such a field exist in the template then it is filled with Articles A. In particular it is possible to list related articles A to an article B based upon matching tags. Finally, articles B are not listed in featured section of index.hml.

While class A articles are best suited for news and more "live" content, class B are indicated for a collection of documents such as a knowledge base, technical documentation or products that are organically correlated.

Pages

Pages are uncorrelated, such as "product", "download", "privacy", and so on. Once you've created a page you can also link statically to it (provided you don't change the title and do not remove the page) in fixed menus directly into the templates, using `{{page_link_<pageID>}}` marker variable.
Pages don't have abstract which is disabled while operating on Pages.

In the index.html page they are listed in a special section (typically a menu or drop-down menu) delimited by the following two markers: `{%xtract:pages_index%}` and `{%/xtract:pages_index%}`. This section is extracted as an embedded template, populated and repeated for each page to list.
Also the marker `{%xtract:page_menu%}` `{%/xtract:page_menu%}` can be used to envelope a portion that is removed if no page is available, for example to hide a button to open an extended menu.

Example:

```
{%xtract:page_menu%}
<li class="buttonmenu" onclick="toggleHiddenMenu()">More pages...</li>
{%/xtract:page_menu%}
{%xtract:pages_index%}
<ul class="hidden"><li><a class="mnu_ix_pgs"
href="{{article_link}}">{{article_title}}</a></li></ul>
{%/xtract:pages_index%}
```

Pages are rendered using *templatepage.html* template.

Furthermore *Pages* may be linked using the marker variable `{{page_link_<pageID>}}` (where `<pageID>` is the ID of the page, which can be copied into the clipboard by clicking on the page title located above the list of pages). For example `{{page_link_products}}` would be a page titled "products". You can use this link in fixed positions into your template, for example to customize your menu.

Page as Index (Entry page)

While BlogManager automatically generates the index pages featuring all non-hidden articles A, and listing all articles B and the Welcome section can be inserted into the first generated index page, it is also possible to mark one Page as entry (main Index) page. Only Pages can be used for this purpose and only one page in the whole project can be marked as entry main Index page.

Once the page is marked as main Index it is used to generate the main index.hml (or welcome.html) as specified in Site Settings and it is removed from the list of regular Pages. A specific templatepageindex.html file is used as well. Unlike autogenerated index pages, the title you entered for the page is used in place of the Site name and Tag line.

This allows to create an entry page that does not show the featured, most recent articles A. This fits cases where a website is not primarily meant as a blog or news site. Still it is possible to link the first index page with featured articles, for example under an item of the main menu of this entry page.

Remarks: If the Page marked as Index is also hidden it is removed from output, and the main entry page becomes the first automatically generated index with featured articles.

No Page as Index	Automatically generated index pages with featured articles A start from default index.hml (or whatever specified in Site Settings). Page title, typically: `{{site_name}} - {{site_tagline}}`
Page as Index marked	The page is made the main entry index.hml (or whatever specified in Site Settings). The first automatically generated index page with featured articles A (and Welcome Section, if filled) becomes index0.html . You can link index0.html into templatepageindex.html . Page title, typically: `{{article_title}} - {{site_name}}`

Hidden documents

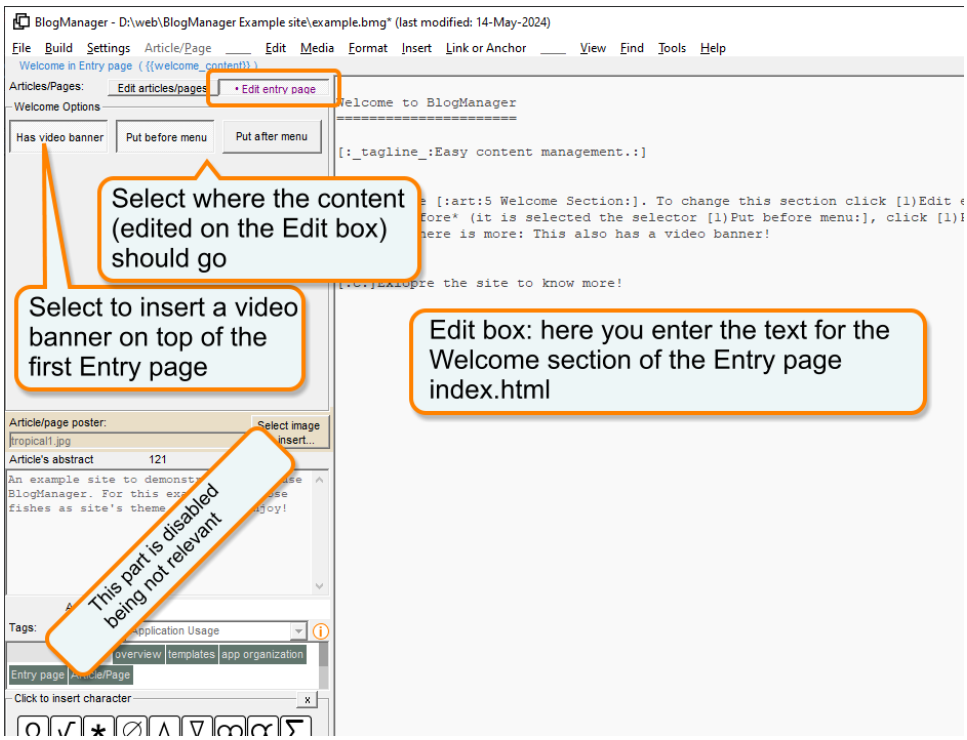
Every document can be set to be hidden, which means it is kept from generation and upload. This is useful for draft or repealed documents. Hidden documents are not deleted from disc, though.
If a document has been uploaded and then repealed making it hidden, **the uploaded files still remain on server!**
However BlogManager try to make all links to the hidden page unreachable redirecting them to the notfound.html page. The pages that are affected by this change are elected to be regenerated - uploaded if they have been changed before last generation.

***Caution*.** If you change a document after the last generation date, and such document links to a hidden document, BlogManager has no way to know whether you intended to keep that link alive or not, so the page may still unintentionally link to a repealed document.

So far BlogManager do not generate a delete command for hidden documents (and their accompanying images or videos, into the CURL batch file. So it is recommended to remove manually the documents and files that should be meant to no longer stay on a publicly accessible webserver.

Welcome section

Besides Pages and Articles A and B, you can also create content that would be shown in the first index page, the welcome section. This content is entered entered into index.hml (or whatever name specified in Site Settings) if no Page as Index is defined, otherwise it is entered into index0.html.



This welcome content is inserted in place of the marker variable `{{welcome_content}}` that in turn could be surrounded by `{%xtract:welcome_content%}` `{%/xtract:welcome_content%}` to embed a template for the welcome content. This embedded template can be removed altogether if no welcome content is available (that is you didn't create its content). This is useful also to remove the embedded template if more index pages are created to list all the articles.

Example:

```
{%xtract:welcome_content%}
<section id="welcome">{{welcome_content}}</section>
{%/xtract:welcome_content%}
```

To switch between articles/pages and the Entry page's welcome content click the buttons on the top left corner or just press F6 to edit Articles/Pages; or F7 to edit the Entry page.

The welcome content can contain any element that can be entered in pages/articles, including images and videos. However sections `-§` are ignored and the text cannot be split in sections as mentioned above for pages/articles, but create embedded sections inside the content. This also means you have a limit of about sixty thousands characters for the welcome content.

Before or After?

You can select where the welcome content goes using the mutually exclusive selectors in **Welcome Options**. There you can also enter a banner video that is used only for the welcome page (the first *index.html* index page). Where banner and welcome exactly go is up to you as you can change this in your templates.

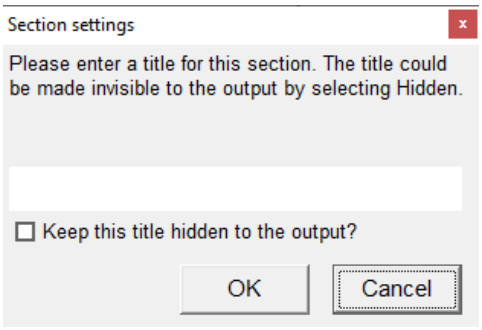
Index pages

A series of index pages are generated to automatically list all Articles A and B and Pages. More accurately Articles A are featured from most recent back to older in a featured section replacing `{{articles_recent}}` with template recentrecord.h .
In addition, and only on the first index generated page, Articles A are listed into section `artlistA` if provided, Articles B are listed into the section `artlistB` if provided, and Pages into `pages_index` section if provided into templateindex.html.

You can set the limit of number of articles to list per page in **Site Settings**. Thus BlogManager will create more index pages to account for all the articles. Each page is progressively numbered, example *index1.html*, *index2.html* ...and so on. Welcome content is included in the first index page only named *index.html* or *index0.html* if a Page as Index is defined.

Entering text - Sections

On the right pane (see picture above) of the application, page/article's text can be entered.
For organizational reasons, and future development of pages with more features and because of limits about the number of characters that can be entered into the edit box, text may be broken in sections. Each section can host no more than about sixty thousands characters. When you are about to reach the limit the character counter on the bottom left corner becomes red. If you have to write more text you can create a new section.



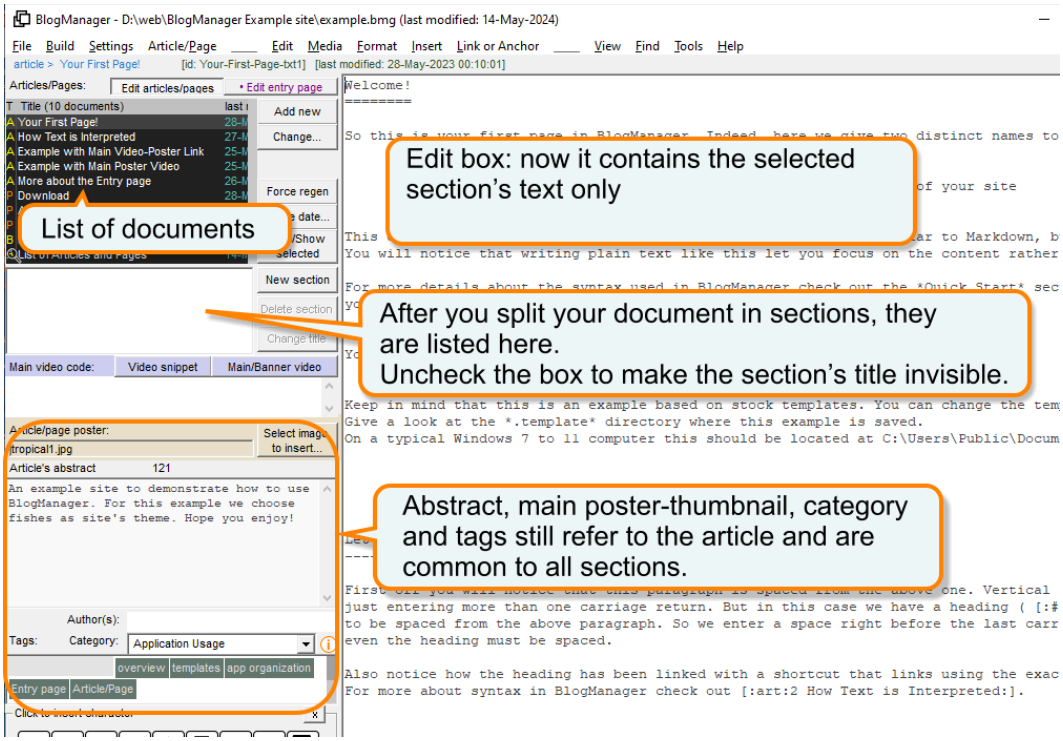
If no section is defined, the whole text fall into the default *section* (which is not surrounded by a `<section>` element when converted to HTML).

To add a new section click `Add section` button.
When almost one section is created, every part of the text

will be surrounded by the respective `<section>` elements.

The Edit box then contains the selected section, while the others are kept in memory and listed in the Sections list.

Each section can be identified by a title, and such title can be rendered as a *heading* (element `<h2>`) for the section, or you can choose to keep it hidden from rendering.



Sections and relative titles (and hidden attribute) can also be created by just entering the specific marker `-§` (see *Markers*), though in this case the section is extracted after the article/page is saved and reopened (example, by selecting to edit another page by clicking on the Articles/pages list).

This allows you to keep your content better organized, and we as a developer to avoid the use of third party edit components, making the app lighter, faster and less buggier relying solely to the Operating System's stock components.

Also this allows to be ready for future developments for richer pages, i.e. pages with tabs.

Generation

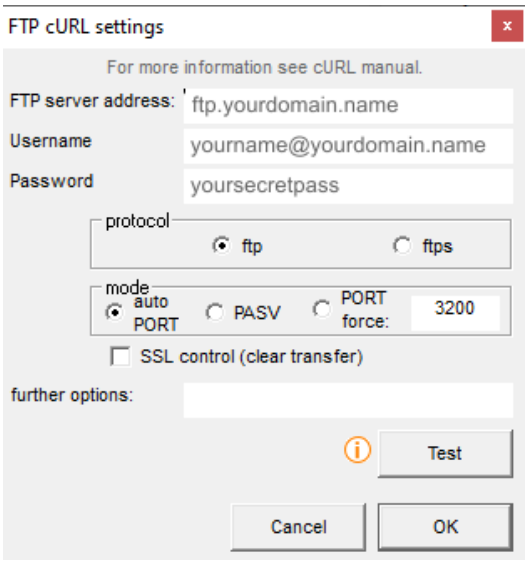
Once ready, you can generate the site, either in full or limited to the last changes. The generation creates a full directory structure that mimics the one that will sit on the web server, so it is easy to copy the files from the local directory to the home site directory maintaining the same structure.

When BlogManager generates the site it produces the entry page (typically named *index.htm*), if necessary adding multiple sub indexes (named index**#**.html, where **#** is a progressive number) and a summary site map both human readable (typically named *sitemap.html*) and machine readable (*sitemap.xml*) and the *robot.txt.* In addition it can genereate a lookup index with the list of tags and the associated articles and categories. This file is generated both in a human (and robots) readable way creating a specific HTML file, and in a machine readable way specifically designed to be loaded into a script, e.g.:, javascript, creating a JSON file, used by the *search.html* page.
In case a Page as Index is defined the main entry page is generated out of this page, and the generated index pages start from *index0.html*.

It also copy a list of file that can be specified into the *rootcompanionfiles.txt* (see Companion files and special folders cmimgs and docs).
All the images included into the pages are copied into the destination directory, mimicking the structure at the server on line.

Upload Files

Once generated the site needs to be uploaded to the online server. To perform this task BlogManager generates a batch file with commands to cURL, which is commonly found on Windows 7 and above. But it available even for earlier versions.
Once generation has completed BlogManager prompts you to choose to upload your files using this batch file and the settings you entered in FTP settings within **Site Settings**.



Alternatively you can use a third party application such as, for instance example, FileZilla. (Please note this is just an indication, we are not affiliated with that product or manufacturer.)

Templates - Installation notes

To manage your website you need to create a folder where all the generated files should be placed, and from where the template files used to build your site should be located. For example let's say you create a folder in C:\MySite that will be the target destination for the generated web site.
This path should be entered in **Site Settings** in the field Path where the generated site will go.

Into this path you will put the *templates* folder where the template files will be placed. To get a fresh copy of template files go to File menu and choose Copy templates... then choose the destination path, in our example it should be C:\MySite . After you confirmed a couple of new folders are created with the path: C:\MySite\BlogManager\templates . From there you can move the folder templates above in C:\MySite and you can start to customize the template files to fit your needs.
If customizing the file you add images (either <img> elements or image attributes), the files for these images should be located in a path that should be entered in the field Path to the repository of the images in **Site Settings**.

The page banner (including the one for mobile device) and your logo should be placed in the *templates* folder.

Finally you need a folder where the documents (articles and pages) edited in BlogManager will be saved, it is recommended to locate this folder in your base site path, so for example: C:\MySite\Articles. This path should be entered in the field Path where articles will be saved in **Site Settings**.

If you use downloadable fonts, enter the path where these files are located in your disc in the field Path to the repository of web fonts in **Site Settings**. Use the marker `{{fontpath}}` in front of the path specified in the URL parameter in your CSS file: this marker will be filled with the path specified in **Site Settings** when in preview or the root (home directory) of your site when generated.

Doing this way you can manage multiple websites, creating multiple template folders, each for every site you want to manage.

Please, refer to the chapter *Template Files* .

Working Paradigm

BlogManager holds the information in a site's project file and separated text files for each article. You can save distinctly each one of these two files, but it is usually recommended to save both. The article's text is saved independently into a file named after the title of the article (the name is canonicalized to conform to the file system conventions), while the article's parameters (modification date, tags, abstract, and other metadata) are stored into the site's project file.
Please note that the site's project file can be saved and opened using the menu File*,* Open site*, and* Save site / Save site as...

Within the text of the article a further *header* and *asides* with the special classes "related", "about" and "summary" can be added. However the *header* element could be redundant if the actual Abstract field is used, and BlogManager will strip it out from the text if it meets the header marker, and will keep the content into the Abstract field.

Accessory Fields

In BlogManager there are specific fields that can be used to enhance the quality and semantic of the site. The data provided in these fields are stored into the site's project file.

Article Abstract

This field is located in a box on the left pane of the application. The abstract is held in the site's project data, for the selected article. Alternatively use the Insert menu *Header* to do the same thing, BlogManager will try to strip out the text when the article's text file is loaded, though.

The abstract will be inserted in place of the `{{abstract}}` marker in the template file for the article's page. You can place this marker everywhere in the template page.
If you want, you may also have the abstract inserted within the article. To do this insert the following marker: `\%{{abstract}}%/` where more convenient.

The text in abstract can be marked as in the article's core text, with some limitations.

Caution: Within the abstract only formatting markers, special characters and few more are allowed. Because no paragraphs `<p>` element is created in abstract, to break a line use the marker `SP_CR` .

Author

Use this field to enter the name of the author (or the authors) of the page. This overrides the name entered in the default Author field in **Site Settings**. To enter more than one author, separate each name with a comma. The first name will be used to fill the `{{article_author}}` field, while the following names will be used to fill the `{{article_coauthor}}` field if found, otherwise it will be used the same `{{article_author}}` field. For this purpose it is recommended to wrap the block of HTML that contains the author fields into a `{%xtract:author%}` selector, so BlogManager can properly extract it and populate multiple fields accordingly.

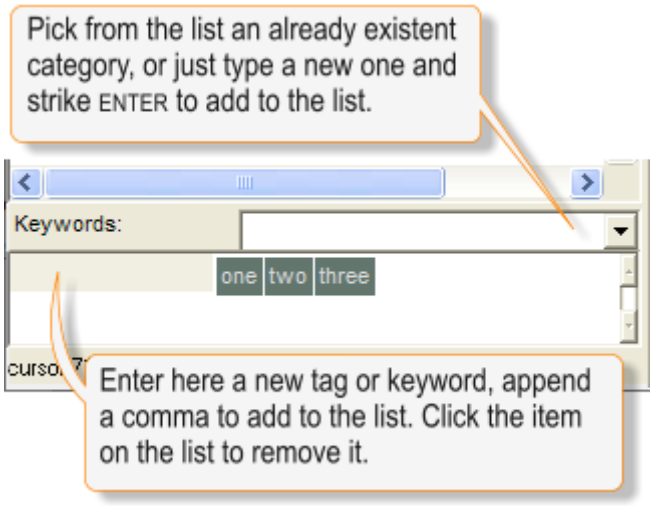
Main video and Poster

Here you can enter the poster of the article or a main video, in various combinations. See ***Inserting Pictures*** .

Tags / Keywords

Into this field a list of tags can be inserted or removed. To enter a new tag, type it into the light-gray text box and append a comma. The new tag will be displayed into the adjacent list. To remove a tag, go over it and click with the mouse.

The tags will be inserted in place of the `{{tags}}` marker in the template file for the article page.
Best place is into the <meta> element into <header> of the HTML file.
But can even place it into the article's text (within the editor) inserting the following marker: `\%{{tag}}%/` .



Category

The category field allows to define a category for the article. The dropdown list will be populated with the categories defined in the whole site project, so before typing a new category you can simply pick up one that already exists from this list just scrolling with the UP and DOWN arrow keys.

The category is inserted in place of the `{{category}}` marker in the template file for the article's page. You can even place it into the article's text inserting the following marker: `\%{{category}}%/` .

Adding a new category

If you can't find any category that fit the article, you can type a new one into the box Category and strike ENTER to add it to the list.

Category removal

To remove a category, select the category from the list and then press CTRL+DEL
Caution: This will also remove the category to the selected Article but won't remove the category from all Articles/Pages that were assigned to that category.

Category renaming

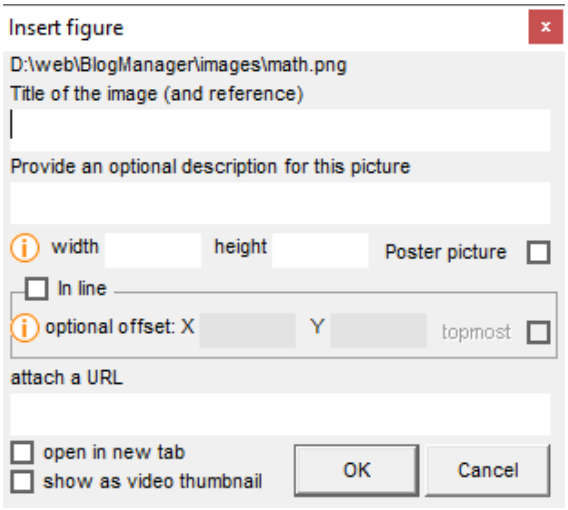
To rename a category, select the category from the list and then press SHIFT+DEL on your keyboard. This will rename the category for the current Article/Page **and for all** the Articles/Pages that were assigned that category.

Note: Both the category and tags are also used to create the Lookup Index and JSON files.

Inserting Pictures

On the left pane the button Select file to insert allows the insertion of pictures into the article. A fully HTML5 compliant element will be generated. It is possible to enter a description, the alternative textual depiction of the image, and optional dimensions (if left blank the image will automatically be redimensioned by the browser). Dimensions must be followed by units, such as px, %, em...

If the Poster picture is checked then the image is not inserted into the article as a regular picture, rather it is inserted as the main *poster* image on top, at the beginning of the article; or it is used as a poster for a video link if a URL is inserted into the Main video field and Video snippet is not checked.



Once inserted, to remove the poster picture just click on the box Poster where the filename is displayed. The mouse pointer changes to let you know this action is possible.

Non-poster pictures are inserted into the article with a comment mark `{!!figure:<ref>!!}` where `<ref>` is the figure's reference ID or the title you entered. This is used to refer to the metadata of the image. If a URL is also provided the image will have an attached link.

The poster image is inserted where the `{{article_poster}}` is located in the article's template or inside the same marker into the file videolink.h if a URL is provided in Main video field.

Remark: Only one poster image per article is allowed.
The Poster image is not the same as the banner and banner mobile pictures (see in *Site Settings*).

Besides poster picture, other images can be inserted as figures and in-line images. The latter can be also positioned relatively to the point where they are inserted.
Normally figures have an *onclick* event attached that opens the image in a new tab, at the widest size possible. Though, if the attach a URL field is filled then the *onclick* event opens the link provided by the URL. In such a case if open new tab is checked the link is opened in a new blank tab.

Images can be used to create a video thumbnail. In such a case the URL should point to a video player (i.e., a link to a YouTube video) and show as video thumbnail should be checked as well. This also causes a play button to appear overlapped to the image.

Inserting Videos

You can have a main video, that either shows up in place of the article's poster picture, or as a video link that is attached to the poster picture, and you can put more videos into the text body.

The main video can be inserted both as a link or as a video snippet. A video snippet is a piece of code that usually includes a player, such as an embeddable video player, or a <video> HTML code. The URL link or the video snippet code can be pasted into the Main video text box.
To tell the difference is the push-button Video snippet.

The videosnippet.h or videolink.h files are in this case used respectively to insert the code or the link at the desired location within the article's template page where the marker `{{article_video}}` is located.
BlogManager automatically deletes the markers that don't fit the combination of selections, so usually you can have both `{{article_poster}}` and `{{article_video}}`. However if you enter both poster image and video snippet of code then BlogManager will include both.

Alternatively the main video can be inserted as the banner of the page. In such a case, if set, the poster picture is still shown in its usual place (likely below the menu), while the marker `{{article_video}}` is deleted. When a video banner is set, the regular banner inserted in place of `{{pagebanner}}` or `{{pagebannermobile}}` is deleted and in its place the marker `{{video_banner}}` is returned. To hide or use a piece of template embedded into the template itself you can use `{%xtract:videobanner%}` and `{%xtract:banner%}` to select what to display in different cases.

Remark: Only one main video link or snippet / banner per article is allowed. The entry page (index.html typically) can host a video banner only.

Main Video options

=> To insert a video URL:

Paste the URL into the text box Main video, and make sure Video snippet is unchecked. To attach the link to a poster picture, click Select image to insert... choose a file and make sure to check Poster .

=> To insert a video snippet (typically a video player) from external sources:

Paste the snippet of code into the text box Main video, and check Video snippet. Poster picture should not be set in this case, if it is set the poster picture is shown alongside the video. If a picture poster was entered, to delete it click on its filename

shown in the Poster box.

=> To insert a video as page banner:

Press Main/Banner video then fill the required fields, and make sure Put as banner is checked. Once clicked **OK** Banner video button remains depressed. This will automatically replace `{{pagebanner}}` and `{{pagebannermobile}}` and render `{{video_banner}}` as mentioned above.

You can enter up to three different type of video sources, they can be either local files or just URLs. Also you can specify a poster that will show up while the video is not yet downloaded, this can be a URL or a local file too. Typical options for a banner video are autoplay, loop while controls should not be checked. Auto play videos are also automatically muted.

If option Put as banner is not checked the video will take place as main video.

Main Video, Video Snippet and Poster COMBINATIONS

⚠ If Banner video is set, Poster is never used and both `{{article_poster}}` and `{{article_video}}` are deleted, and the following points do not apply.
Poster may be used if Banner video is set.

If Banner video is **not** set, the block enclosed by `{%xtract:banner%}` and `{%/xtract:banner%}` is removed. See BANNER VIDEO below.

1. If Poster is set, `{{article_posterURL}}` and `{{article_poster_depiction}}` are always filled, deleted when Poster is not set.

2. If Poster is set and Main/Banner video is **not** set `{{article_poster}}` is filled with the content of postersnippet.h, and `{{article_video}}` is deleted.
However if Main video field is also set with a URL (to a video) `{{article_poster}}` is deleted and `{{article_video}}` is kept and filled with postervideosnippet.h.

   => `{{article_video}}` is always deleted if no Main video is set.

3. If Main video is set or the the field is set with a URL or a snippet player:
   => `{{article_poster}}` is always deleted when Main video or URL/snippet is set.

   - if Video snippet is checked as well: `{{article_video}}` is filled with videosnippet.h which in turn got `{{article_videosnippet}}` filled with the content of Main video field that is supposed to be a snippet of HTML code (i.e. a video player).

   - if Video snippet is **not** checked:

`{{article_video}}` is filled with videolink.h which in turn got `{{article_videolink}}` filled with the content of Main video field (be it a URL or snippet player), then (within videolink.h):

- if Poster is **not** set: `{{article_poster}}` is filled with with "*Watch video*".

- if Poster is set: `{{article_poster}}` is filled with postersnippetvideo.h which in turn got `{{article_posterURL}}` filled with the content of Poster and `{{article_poster_depiction}}` with the content of the description given for Poster (if any).

Please note that the same variable names (i.e. `{{article_posterURL}}`) may be filled with different data when they are in the main template or in an including file, such as postersnippetvideo.h, since it is within a specific context as described above.

REMARKS

Because one of the two marker variables are deleted when building the pages, it is advisable to put both in the same place in your templatepage.html and templatepost.html.

When building *Recent* and *Related* records only Poster is used, even if either Main video field is set (a snippet player or URL) or Main Video is set. It is therefore recommended to always set Poster, even when Main video / Main video field is set.

BANNER VIDEO

In the specific case where Banner video is set both `{{article_poster}}` and `{{article_video}}` are deleted.
In addition the block enclosed by `{%xtract:videobanner%} {%/xtract:videobanner%}` is extracted, while the block `{%xtract:banner%} {%/xtract:banner%}` is deleted.
If the extracted content has the variable `{{video_banner}}`, then that is filled with the video settings entered for Banner video, otherwise the extracted block is replaced with its content as is.

In the case the above condition is not met, namely a Main video is set, then the block is removed altogether, and the content of `{%xtract:banner%} {%/xtract:banner%}` is extracted and put in place of this block.

This provides a mean to switch between a standard, picture based banner, and a video based banner.

Example:

```
<body><div id="PageBanner">
{%xtract:videobanner%}{{video_banner}}{%/xtract:videobanner%}
{%xtract:banner%}
<img id="banner" style="width:100%" src="{{pagebanner}}"
class="hide_small">
<img id="bannermobile" style="width:100%" src="{{pagebannermobile}}"
class="hide_medium hide_large">
{%/xtract:banner%}
</div>
</body>
```

Remark: This operation is performed before the content of the article is inserted into the template.

Videos in the article's body

To insert a video into the article or page body, move to the location in the text where you like to have the video, now from menu Insert click Video... or just press CTRL+M as a shortcut. Fill the required fields and options in the wizard dialog box. Typical options are controls checked, while autoplay not checked.
Once ready click **OK**: a reference like `{!! video:title !!}` is inserted into the point where the cursor was located into the text. You're done.
If you want to review your settings for the video, move the cursor on the reference text and press CTRL+ALT+M or choose menu Edit > Edit video metadata .

ARTICLE OR PAGE IMAGES AND VIDEOS

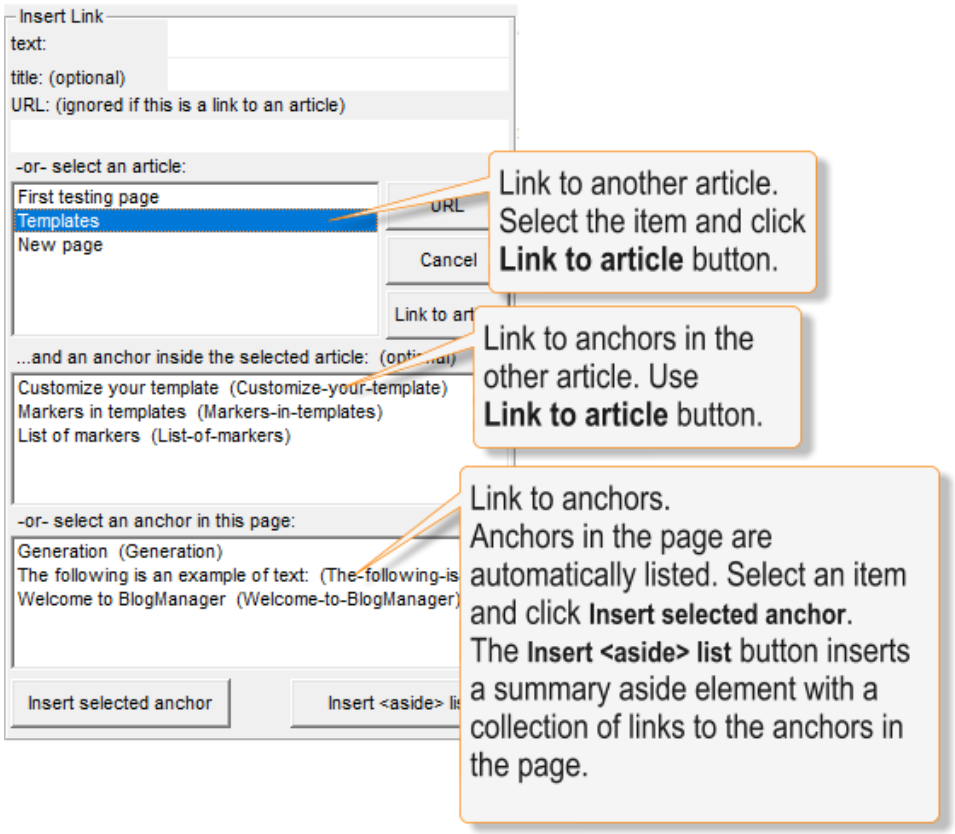
Images and videos (including poster) inserted into an Article (or Page) are automatically copied to the destination path when the site is built. Similarly the images for <img> element in the template are scanned to be copied to the destination path from the template path (and its descendant).
Where the src attribute in <img> or the filename entered begins by one of the following: http:// file:// https:// data:image and {{ , then the file is never copied.

In those cases (so including the cases where `{{relativepath}}` is added at the beginning), the files should be listed into the *rootcompanionfiles.txt*.

Inserting Links

Links can be inserted either by directly typing the markers (see *Markers*) or by choosing the Insert link menu or press CTRL+L. A dialog box make it easy to enter the link or choose a link to another page, or inserting a link to an anchor in the very same page.

Selecting an article the related available anchors are listed below, while at the bottom are listed the anchors available in the same article or page. To link to a page click only the article. To link to a specific position in that page click also the anchor.
To cancel the clicked anchor just click again on the article, the anchor list will be deselected.



Type links manually

If you want to reference another point in the same page (an anchor), let's say a paragraph with heading, you can type a shortcut link in the form of [:#: Title of the heading :]. The link ID is taken from the text you entered inside the block marker `[:#:` and `:]` .
Caution: The text should be exactly the same of the anchor, example the heading, you want to link. Spaces before and after text are ignored, so the following are equivalent:

```
[:#: My Title :]   [:#:        My Title      :]   [:#:My Title:]     [:#:My Title      :]
```

Site Settings

By clicking `Settings` on the menu, the settings pane shows up. Here are located both the site's information.

Site name

Is the name or title of the site. This name is used in the <meta> if the `{{sitename}}` marker is found into that tag. It is also used to name the directory where the generated site is saved. **Required**.

Site keywords, tag-line and description

Comma delimited list of keywords general for the site; a tag line for the site, such as "BlogManager Static site management system"; and a description.

URL

This is the root URL of the site, typically the domain name. **Required**.

Entry page

This is the name used to create the main entry page, the one that the server will return by default. Typically index.htm or welcome.htm . **Required**.

Subsite path

If the site is a subsite, enter in this field the path where the subsite is located (on server side) in relation to the root of the main website. Example. If you have a main site that is located at the root of your server (i.e.: httpd-public) and you want to create a subsite located at a sub directory mysubsite/new (i.e.: httpd-public/mysubsite/new) enter in this field: `mysubsite/new` .
This field should be left blank if you are editing a site that will go at the root of public documents on your server.

Sitemap page

This is the name used to create the map of the site. Typically sitemap.xml.
Required. BlogManager can generate a valid XML sitemap. ***Caution***: BlogManager automatically generates also sitemap.html based on templatesitemap.html.

Path to the repository of the images

This field lets specify where the default images should be picked up. This is typically used for common images, in your local computer. In the body of the pages (and welcome content) you can also enter images from wherever you like from your local computer. No network drive are supported, though.
It is recommended to keep this path under the same folder where your project is saved, such as "images". If this field is left blank BlogManager automatically uses the project's path.

Path where articles will be saved

This field specifies where the text files of the articles will be saved. It is recommended to keep this path in the same folder where you save the project. If this field is left blank BlogManager automatically uses the path where the project file is saved.

Path where the generated site will go

Enter here where the files will go when generating the site. **Required**. If not specified BlogManager assumes the same location where the project file is saved. It is recommended to enter a name such as "generated".

Copy images

Select this option to copy the picture files from the source folder (the one specified for the repository of the images, or wherever else you picked them) to a folder specified in the adjacent text box (relative path). Required.

Generate a directory per each article

Select this option to save distinctly in specific folders the pages generated for each article and the companion images (see also: *Companion files and special folders cmimgs and docs*).
If this option is not checked the **required** Server side subfolder for articles should be filled, and it will be where all the generated pages will go. This option is not recommended though.

Author name

Enter the default name of the author for articles and pages. This name is used if nothing is entered in the same name field for the article/page located on the left pane of the application. Either this value or the one specific for the page/article will be inserted where the marker `{{article_author}}` and `{{article_coauthor}}` is found.
If more than one author have to be entered, separate each name with a comma.

Email Addresses

Here you can enter a list of email addresses, each one must be followed by a reference name, separated with a semicolon. Example: *john.doe@example.com;sales-manager*. This reference name should be used in the marker variable `{{email: }}` that would be replaced with the related email, example: `{{email:sales-manager}}` . To add the email address typed into the box press ENTER.
Remark. This is provided for future versions of the program, in this version email addresses are collected and saved only, they do nothing and the related marker variables won't be filled (so do not use them for now!)

Max number of recent articles to list

- per index page : Max number of articles to list per each index page. If more articles are available they will be placed in further index pages. All index pages are automatically linked together.
If most recent by last modified date is checked the most recent is taken using the last modified date, otherwise it is taken using the creation date (not the creation index).

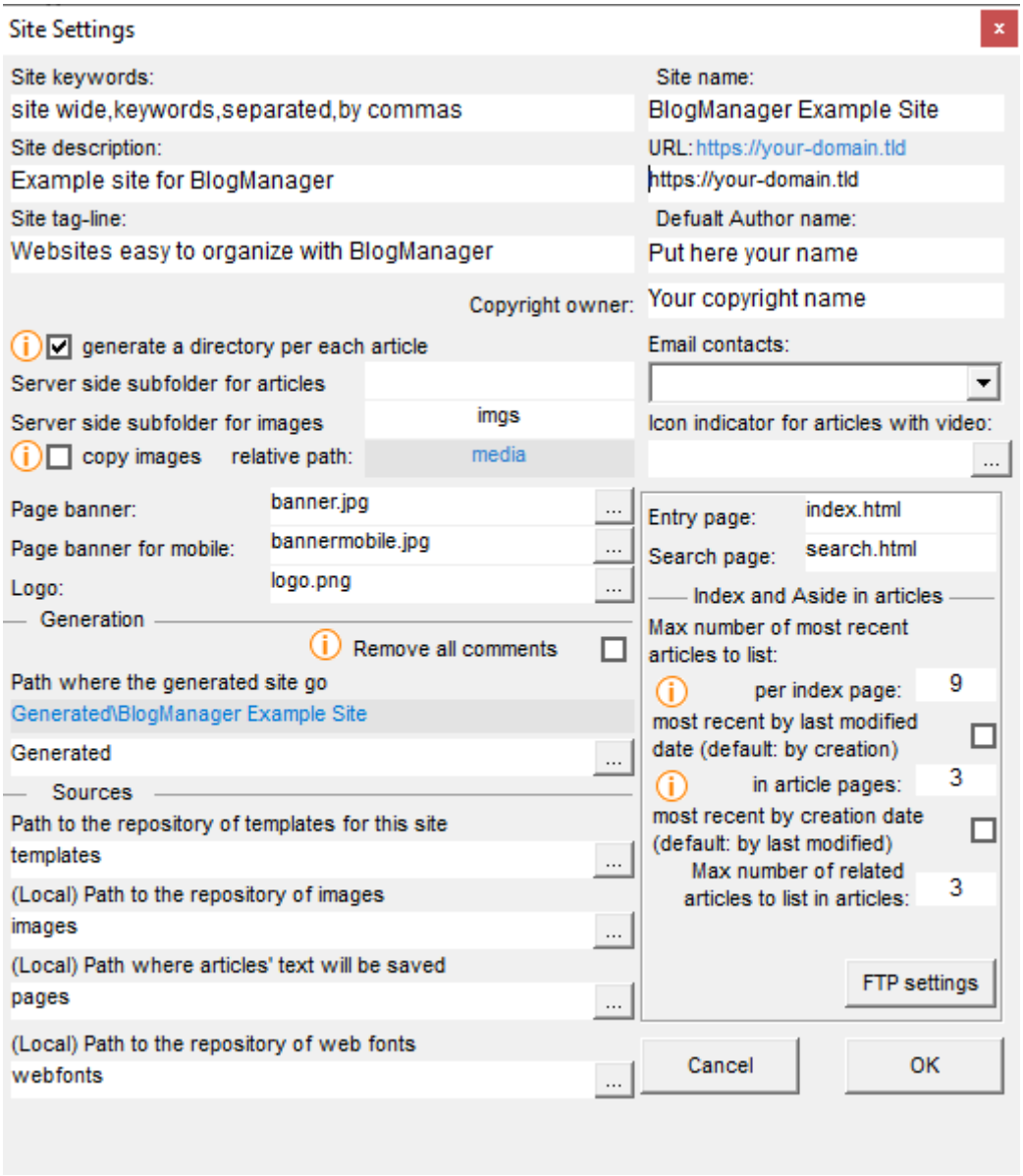
- in article pages : Max number of most recent articles to list on the related aside in article's pages.
If most recent by creation date is checked the most recent is taken using the creation date, otherwise it is used the last modified date.

Max number of related articles to list

Set the maximum number of related articles to list on the related aside in article pages.
Related articles are picked counting **the number of tags** *(keywords)* **that match** with the current article.

Icon indicator for articles with video

Enter the filename of an icon that will take place of `{{article_hasvideoicon}}` if the article has almost a video.


Site Settings
Site keywords:
site wide,keywords,separated,by commas
Site description:
Example site for BlogManager
Site tag-line:
Websites easy to organize with BlogManager
Site name:
BlogManager Example Site
URL: https://your-domain.tld
https://your-domain.tld
Defualt Author name:
Put here your name
Copyright owner:
Your copyright name
generate a directory per each article
Email contacts:
Server side subfolder for articles
Server side subfolder for images
imgs
copy images
relative path:
media
Icon indicator for articles with video:
Page banner:
banner.jpg
Page banner for mobile:
bannermobile.jpg
Logo:
logo.png
Entry page:
index.html
Search page:
search.html
Index and Aside in articles
Max number of most recent articles to list:
per index page:
9
most recent by last modified date (default: by creation)
in article pages:
3
most recent by creation date (default: by last modified)
Max number of related articles to list in articles:
3
Generation
Remove all comments
Path where the generated site go
Generated\BlogManager Example Site
Generated
Sources
Path to the repository of templates for this site
templates
(Local) Path to the repository of images
images
(Local) Path where articles' text will be saved
pages
(Local) Path to the repository of web fonts
webfonts
FTP settings
Cancel
OK

Application Settings

General section: Reload on start up

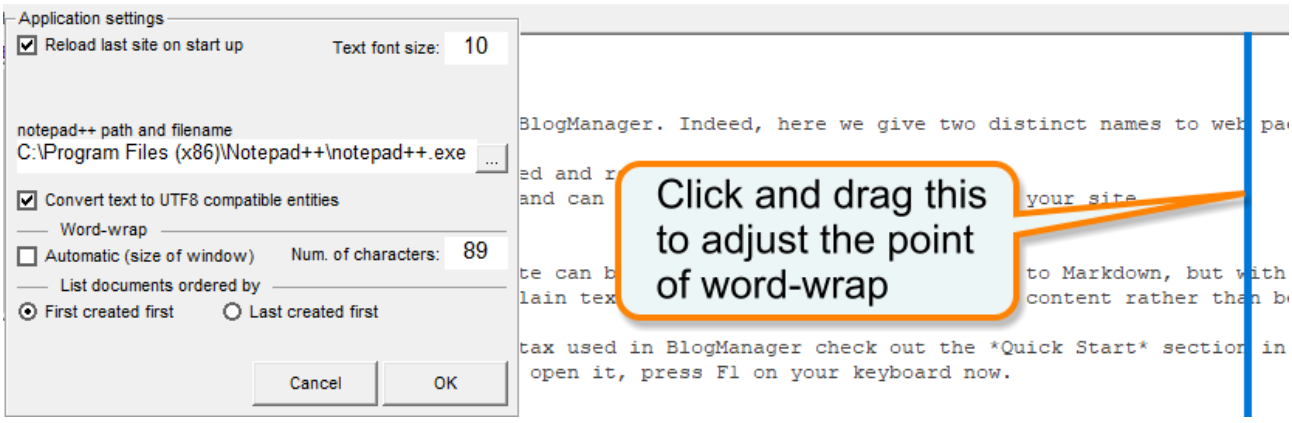
This is general, common for all sites. Specify that BlogManager will automatically load the last opened site.

Notepad++ path

Optional. Specify here where the program Notepad++ can be located to use it as an external editor or inspector for text of articles.
Please note that editing the articles externally may cause some features to be lost. The button `Add from file...` can be used to add external files to the project.

General: Font size

Specify the size of the font for the text editor on the right pane of the application.



Word-wrapping

Enter the number of characters you'd like to use to define the width of the lines either when auto word-wrapping is activated (press `F9`) while typing or to word-wrap the selected text by pressing `F11`. If `Automatic (size of window)` is checked then the number entered is ignored and the width of the editing box is used instead.

Ordering

Articles and pages are listed by order of creation. You can choose to list by first last created or first first created. Note that the order refers to the actual creation, not the creation date as listed which could be modified by `Force date`.

Notes on Character Set

The generated pages are based on Windows cp1252 codepage, which is accepted as ISO-8859-1, however all the characters above 0x7F (127) ASCII are converted in character entities or named entities, example the character È (0xC8 (200)) is converted in È so the resulting page is compatible with both UTF-8 and ISO-8859-1.

So both headers <meta charset="ISO-8859-1"> or <meta charset="UTF-8"> are valid.
However data supplied in attributes of HTML entities such as the attribute description of the <meta> entity could be read improperly if UTF-8 is used and non basic ASCII characters are used.

The option **Convert text to UTF8 compatible entities** under **Application Settings** menu allows to enable or disable the automatic conversion. When disabled the output HTML is not converted and ASCII extended characters (those in the range 128-255 (0x80-0xFF)) will be returned as they are.

Unicode

Unicode characters are not supported. To insert a Unicode character you can enter the code prefixed with \&# example: `\Ӊ` notice the space before the backslash. Similarly named entities can be inserted, example: `\φ` which is rendered φ .

List of Articles and Pages

On the top of the left pane is located the list of the articles and pages, and the button Add article, and Change... (more options are available under menu Article/Page) .
The list shows the title of the article or page, creation and last modified date. On the left is shown the type or status: A or B means article class A and B respectively, P means page, and H means hidden.

A hidden document means that it is kept from output generation, rendering, indexing, lookup index and from sitemap.
Hidden status is useful for draft or repealed/archived documents.

Clicking on top of the list, on each specific column, it is possible to view only one type of document, reorder by title on acending/descending alphabetic order, reorder (ascending/descending) by last modified date. Clicking cyclically change the reordiering and when no symbol (or dot) is shown the list is ordered by the creation order (either first last created or first first created accordingly with the **Application Settings**).
Click the magnifier icon to enlarge the list.

Selecting an article it will be displayed into the editor, on the right pane.
To save an article either click on the File > Save article menu or press CTRL+S, or just save the project site through File > Save site or pressing CTRL+W.

Document attributes

Menu Article/Page offers a list of all possible attributes that you can change. You can also right click on a item on the list of articles to achieve the same, contextualized menu.

- Button Change... allows to change the title and other properties of the document.
- Button Hide/Show allows to switch the status between hidden and visible of the selected item. Also see remark below.
- Menu Article/Page>Remove remove command. When a document is removed its content file is not deleted and remains on disc. However all metadata and extended information such as the collection of videos and images that are stored in the project file are deleted.
- Button Force regen tells BlogManager that the selected article or page must be regenerated regardless of its last modified date. Regenerated documents are elected for upload as well. Once the item has been forced it is listed on a red background. Clicking again the same button clears this status. Forcing regeneration is not saved and it is automatically cleared once the program is closed or once the site has been built. Also forced regeneration and upload happens for those documents

that links to a hidden document.

To change the type of document from Page to Article or vice versa use the menu Article/page > Change type. Alternatively click button Change... to view and set properties with a single step.

- Button Force date... allows to set an arbitrary date for the last modified and created time stamp of the document. Note that this affects only the project's metadata and does not change the time stamp of the file on disc. Also this does not affect the creation order.
Caution: Differential build (Build modified since last generation) would skip documents that have been forced to a previous date since last generated date. Use Force regen to tell the program to regenerate the document.

Remark on hidden documents

Hidden documents are not generated nor enlisted for the upload. If any page/article links to the hidden document, that page or article is forced to be regenerated and uploaded.
However this happens *if and only if* the document that has been made hidden is changed after the last generation **and** the page that links to it was changed before the last generation.

List of Sections

This list is located below the list of the articles/pages. By clicking on a listed title the relative section is loaded into the edit box, while the incumbent text is saved into the previously selected section.
The New section, Delete Section and Change title buttons are self explanatory. The check boxes in the list allow to toggle between hidden and visible title, ticked means hidden.
Note that hidden title means that only the title of the section is hidden, not the section.
Non hidden titles are rendered as `<h2>` elements in the final generated HTML, while automatic headings (those marked by a underline made with a series of ==== or -----) are demoted from `<h2>` to `<h3>` and from `<h3>` to `<h4>`, respectively.

Remarks: If no section exists, the text will not be inserted into any `<section>` element.

See also: Editing sections.

Other application menus and functions

View

The menu View > output opens a text box, on the left pane, where the encoded output of the article, or of the abstract (whichever was selected at the moment the view menu is clicked), is displayed.
Use F4 on your keyboard as a shortcut to this function.
It can be used to inspect the output in relation with the input text.

Use CTRL+F4 to enable error checking. When enabled while typing the text is validated, and if any error is found a warning sign appears on top right corner of the app. By clicking on it, the cursor is positioned where the error was found in the text.

Under the view > preview menu a preview of the article is shown into the default browser. Use F5 as a shortcut for preview. Please note that the page shown will not have any related and recent articles listed, but the very same article is shown.
In the case the document is a page the output is rendered using the proper template.
If the current selection is to edit the Entry page, the preview shows an example of the index page, but with no articles listed.

If specified, it is also possible to open the source text file into notepad++ by selecting the related item into the view menu.

Find

The menu Find>Find-Replace (shortcut: F3 or CTLR+F) opens the classic dialog box to search or replace for a specific chunk of text within the selected edit box.

The Find>Find by content, category and tags... (shortcut: CTRL+SHIFT+F3)allows to find articles by tag or category, title and content within the whole site project.
Please note that if the current document was changed you will be prompted to save changes.

Searching in content will also search in titles, section's titles, abstract, and content of each section (if any) or the main content if the document has no sections.
The result is shown on a list that appears at the bottom of the editor pane. Clicking on each line brings to the found content.

Build

The Build menu, generates the site. It is possible to operate in four modes:

- Delete All and Build — Deletes all the files in the target generation directory
- Build all — Generates all the required files into the target directory
- Test build — Similar to Build All but generates files in test mode, Excluding code wrapped in `{%xtract:hide_production%}` and saving site in a special folder _test_ under the generation folder specified in **Site Settings**.
- Build modified since last generation — Generates only the files that have changed since last time the generation was performed. CAUTION: All articles or pages whose last modified date pre-dates the last generated date will be skipped, including those you changed but **forced** to a prior date (see Force date... see also Force regen).

Format

Allows to insert styles to the selected text.
Enumerated and Named styles are picked from the site.css file.

Insert

The Insert menu allows to insert a marker where the cursor is located within the text. For special characters the F12 shortcut is provided.
The item video... (shortcut CTRL+M) opens the dialog wizard to insert a video where the cursor is located. See chapter Inserting Videos .

Link or Anchor

Menu Link or Anchor > Insert Link (shortcut F12) inserts a link where the cursor is located, providing a wizard that let you insert text, title and link easily, also selecting sections and the auto-generated anchors.

Menu Link or Anchor > Anchor inserts an anchor where the cursor is located.

Word-wrap

Under Edit menu auto word-wrap (shortcut F9) toggles automatic word-wrap while typing when the line of text reaches the end of the visible width of the page.
Notes: Hard breaks are inserted, so the text remains unaltered even when resizing the application window. Word-wrapping is prevented if the current line of text have markers that may lost their meaning if they are not found at the beginning of a line, and if the line contains tabulation characters that have special meanings (text indentation or creation of tables, see related chapters).

Menu item Re-word-wrap paragraph (shortcut F11) rearrange the selected paragraph to word-wrap again for the current width of the visible text, removing previous breaks and inserting new ones, if necessary. The function gives a warning if the selected text contains tabulation characters or dividing markers (e.g. -§ section marker), offering the chance to remove all them automatically, for the selected text, or to cancel the operation.

Caution: Re-word-wrapping causes the removal spaces and carriage returns/new lines (CR+LF) that are properly moved at new locations. Multiple spaces or CR+LF are removed altogether.

Mathematical Formulae and Equations

Since markparser version 1.0.4.23 it is possible to enter math equations using a similar syntax of LaTex. A subset of commands is made available, and some semplifications have been made.

Mathematical expressions must be enclosed by $$ and $$ for a display block, while (and) can be used for an inline expression. Note: $$ or)$ can both be used to terminate the math block.

The \begin \end commands are replaced with direct commands: \matrix \array and \table.
An optional argument within brackets can be used to specify the style of the \matrix, \array or \table. Example: \matrix[bm_hatch]{ a & b \\ c & d } specify a matrix with internal lines of separation. These styles can be personalized as they are defined into the bm_styles.css file.

In addition to the above mentioned commands, the following commands are available:
*argument***limits**_{ } ; \frac{}{} ; \binom{}{} ; \ceil{} ; \floor{}; \sqrt[]{} ;
\sum_{}^{} ; \int_{}^{} ; \oint_{}^{} ; \iint_{}^{} ; \iiint_{}^{} ; \prod_{}^{} ;
\text{} ; \hat{} ; \bar{} ; \overbar{} ; \overrightarrow{} ; \overleftarrow{} ; \vec{} ; \mathrm{} ;
*argument***supstack**{} ; *argument***substack**{} ; \matrix{} ; \array{} ; \table{} ; \left. \right.

\left and \right are available but an alternative syntax with simple braces is recommended.
Example: `{ \sqrt{ a + b } | }^2_n` produces: $\sqrt{a+b}\Big|_n^2$

Furthermore the following commands are available:
\lim \exp \bmod \cos \sin \tan \cot \sec \arccos \arcsin \arctan \arcsec \sinh \cosh \tanh \coth \log \ln
\to \infty \times \bullet \dots \vdots \cdots \ddots \propto \therefore \because \sim \simeq \neq
\summation \integral (alternative use of \sum and \int) \leftarrow \uparrow \downarrow
\langle \rangle \doublebar \| \& \product \parallel \nparallel \oplus \ominus \otimes \oslash \odot
\real \pm \mp \div \Uparrow \Downarrow \implies \iff

Display property

Instead of \display use \Sum \SUM \Prod \PROD and \Int that place the following arguments below and above the symbol. Remark: they must be used inside a block section, otherwise they will be considered as non-display. If \Sum and \Prod are used inside a \frac or \binom or other \sum\int... the display style is automatically turned off by some browsers. To avoid this use \SUM and \PROD. Do not use \SUM and \PROD outside \frac \binom, etc.

Examples:

renders as:

```
$$ T = \Sum_{i = 0}^{2^2n -1} data_i $$
```

$$T = \sum_{i=0}^{2^{2n}-1} \text{data}_i$$

```
$$\text{scaled data} =
  \frac{\ceil{\SUM_{i = 0}^{2^2n -1} data_i}}{2^n}$$
```

$$\text{scaled data} = \frac{\left\lceil \sum_{i=0}^{2^{2n}-1} \text{data}_i \right\rceil}{2^n}$$

```
$$\text{scaled data} =
  \frac{\ceil{\Sum_{i = 0}^{2^2n -1} data_i}}{2^n}$$
```

$$\text{scaled data} = \frac{\left\lceil \Sum_{i=0}^{2^{2n}-1} \text{data}_i \right\rceil}{2^n}$$

Engineering Extensions

*argument***updw**{ *up* }{ *down* }

Can be used to place two arguments, one above and the other below the previous *argument*.

u{*argument*}

Specify a unit, and optionally a metric prefix. Example: `350\u{Wm}^2` results in: 350Wm^2

Special symbols for electrical engineering: \DC \AC \GND \electbolt

Remarks and Variants

Some pecularities and differences in respect to LaTex syntax:

- Three consecutive spaces are detected as a space separator. Add more spaces and the space separator becomes wider.

- Alternative **display**. Expressions that begin by $$ have the display attribute set, and are shown on their own block. Expressions that begin by $(have the display attribute set to false, and are shown in-line with the text.
Integral, Summation and Product symbols are shown as if they were with the display attribute set to true irrespective of the block style by using commands \Int \Sum \Prod (first letter capitalized). However some browsers may override this for \Sum and \Prod, to force display in such a case use \SUM and \PROD (see Display Property.)

- Discouraged use of **left** and **right**. Wrap expressions in braces instead. This result in a easier to read and more compact script. Put parentheses outside the block to make them wrap the whole block.

Example: `x = {(\frac{a}{b})}` produces: $x = \left(\frac{a}{b}\right)$

- ***Caution*** with tabulations. While you can use tabulations inside a math or chem expression, tabulations are not allowed inside tables. So expressions inserted into a cell of a table should not embed tabulations otherwise an error occurs.

Chemical Expressions and Formulae

Since markparser version 1.0.4.23 it is possible to enter chemical expressions.
As for math expressions the expression must be enclosed by `$$` and `$$` for a display block, and `$\` and `$$` for an inline expression. All math commands can be used within chemical expressions, with some limitations. However inside chemical expressions a leading and trailing space is required for commands that has arguments.
A chemical expression must be provided as argument of the command `\ce{ argument }`.

Special commands:

```
\lrarrows \dblharpoon \lrarrow \udarrow \tricolon
\bond{-} \bond{=} \bond{#} \bond{~} \bond{~-} \bond{~=} \bond{-~-}
\bond{...} \bond{<-} \bond{->}
```

These commands do not allow any space when inserted between chemical elements.
In addition no spaces are allowed inside braces for bonds. For instance `\bond {-}` is illegal.

Greek letters

Inside chemical expressions greek letters can be embedded in between chemical elements, however a semicolon should be added to terminate the command for displaying the greek letter before a new element begins, or the command must be enclosed in braces or parentheses.
Merely illustrative examples: `NO\tau;H2  Py_(\sigma)  R{\alpha}`

Chemical Elements

Within chemical expressions a special syntax is available to fit chemical elements:
Elements:

- They must be defined with letters, no distinction is made for upper or lowercase. An element can be followed by another element.
- Basically it is treated as a string of text as long as only A-Z and a-z letters are found and the symbol @ is allowed as well.
- First letter must be uppercase. Lowercase leading letters are ignored and discarded. Maximum two characters.
- A number that follows a letter is meant as the number of atoms in a molecule or chemical compound, a following sign or dot or star is meant as the ionization or excited state or a radical.
- Sign or dot or star can follow the letter in place of a number.
- See also post sub- and super-scripts below.

Examples:

```
^235U   KOH    SO4^2-   BA2+    BASO4_(s)
```

prefixes:

- ^ and _ can prefix an element to specify atomic number of nucleons (or mass number A) and proton number or atomic number Z respectively;
- numbers used in these prefixes can be preceded with a sign and terminates at the first non-number found.

- ^{ .. } and _{ .. } prefixes as above, but they can contain any kind of text such as number, signs and letters. Useful for certain isotopes whose number ends with a letter such as ^{99m}Tc .

post sub and superscripts:

- ^ and _ can be used to specify a super and subscript for the element. These are recognized only after one or more letters (see Elements above) and admit only numbers.
  If a number must be followed by a sign, like in the case of the excitation state of an atom or in the case of ions, then the value must be enclosed in braces.
  Example: `Cl^{-}` Notice though that simple expression can be just written as: `Cl-`

- `^{ .. }` and `_{ .. }` postfixes can be used for numbers, signs and letters mixed.

Example: `H2^{4+}_(g)` or equivalen: `H_2^{4+}_(g)` .

- `_( ... )` this is a special postfix that is allowed only after an element to specify the state of matter (liquid, gas, ...) (see example above). ***This component must be the last part of an element.***

Variables

Variables can be wrapped within two dollar signs: `$x$`
If specified, this part must be right after the element. If located after the exitation state it produces an error. Example: `NO_$x$`

Commands and Constant names embedded

Within a \ce command any other mathematical command and any other constant such as greek letter names (i.e.: \phi) are allowed, even ricorsively. No other \ce command is allowed though.
However command \math is provided to allow embedding math expressions inside a chemical expression and command \mi to insert a mathematic identifier.

Preventing ambiguity

In some cases it could be difficult to tell if a superscript belongs to the previous or the following element, such as in `H_64Gd` may mean 64 atoms of H or the atomic number Z of Gd. Or if a subscript is a stoichiometric value.
To avoid this cases of ambiguity a space or pipe (|) can be inserted in between, like:
`H _64Gd` or `H|_64Gd` this separating character is not rendered to output.
Operands such as + and - should always be isolated with at least a leading and a trailing space.

Parentheses in formulae

Use the same rules seen for Math.

Precipitates/gases

Isolated (with leading and trailing space) character '`v`' or '`(v)`' indicate a precipitate (a downward arrow); a isolated '`^`' or '`(^)`' indicate a gas (an upward arrow).
Important: This is recognized only inside a \ce command.

Reactions

Use `-> <- <--> <-> <=> <=>> <<=>` to specify reactions. Append `[ ... ]` to show a text or formula above the reaction arrow, append a second `[ ... ]` to show text or formula above and below the reaction arrow.
Examples: `->[\text{heat}][CaCO]  ->[ [ H2_(g) ] ]`
To embed brackets or parentheses inside the bracketed block make sure a space lead and trail the bracket/parenthesis.
Remarks: The "arrows" must be separated by almost one space. No space is allowed between the arrow and bracketed arguments, though.

Under set, over set and under-over set arguments. Use the same math commands: \substack \supstack (or \superstack) and \updw. These commands require to be attached to a previous argument that you can provide by wrapping multiple arguments in braces. Example:

`{Zn(OH)2 v}\substack{\text{amphoteres Hydroxid}}` render as: $\underset{\text{amphoteres Hydroxid}}{\mathrm{ZnOH_2}\downarrow}$

Stoichiometric numbers and variables

A number that lead a letter is detected as a stoichiometric number, in case the stoichiometric number may be confused with the trailing number of atoms or molecules a space or pipe (|) che be used to explicitly delimits atoms/molecules.
Stoichiometric variables can be defined by wrapping it with `$` signs: `$n$` Example: `PO_$n$`
Variables can be any a-z A-Z letter or any macro name like \tau.
Macro names are considered variables even if not wrapped in between two $ signs.

Unrecognized entities inside \ce command

Math a-z variables or identifiers are not recognized inside the \ce command, since it is expected to meet only chemical symbols. However all commands are recognized.

To insert a math variable inside a \ce command, use command `\mi{}`.
Example: `\ce{ \mi{a} + \mi{b} = \mi{c} }`

To insert a whole mathematical expression inside a \ce command the command \math is provided. This command disables the chemistry interpreter and re-enable the math x-Tex interpreter.
Example: `\ce{ R <- \math{ \text{components}   a + b } }`
Render as: $\mathrm{R} \leftarrow \text{components } a + b$

Markers

Writing articles with BlogManager is easy, with the help of simple markers the article is both easy to read and clean in its essence. The markers are small symbols that fit for good readability while maintaining the semantic elements of the text.

Delimiters

The following list of markers allow to delimit the text into semantic sections whose types are: Header, Section, Aside and Footer. Opening a new section automatically closes the previous one, if any was open. No sections can be defined within a table (see *Tables*).

marker	example	description
The following markers are recognized if placed at the beginning of the line only.		
`-*`	`-****-` `\-* not recognized`	**Create header** *Note*: not to be confused with *heading*. Any number of asterisks, final dash optional. Header of the article: everything following this line is included into the <header> element. ***Remarks***: This marker should be avoided if the template already has a header that is used to host the article's abstract. A new line of the same type or marking a new section will automatically close the <header> element. No more than one header should appear in one article (the whole content).

marker	example	description
`-=`	`-=====-` `\-= not recognized`	**Create a section** Dash, followed by any number of equal signs, final dash is optional. Create a new section. A new line of the same type will close the <section> and will open a new one. Any other non-nestable element (such as aside, footer, header) closes automatically the section. **It is recommended to use the Section tools in BlogManager and avoid manual entries though.**
`-§SP[title]`	`-§ New section`	**Create a section with optional title** This creates a section that is stripped out into a new section that is listed in the left pane. *title* is optional, if given the title will be used to create a heading <h2> and an anchor id with the same name but spaces and non-ASCII characters are converted into underscores, additionally the length is cropped to 100 characters. Automatic headings inside a section are demoted from h2 and h3 to h3 and h4 respectively for underlined text with `====` or `----`. Remarks: If the title is prefixed with '!' the title is hidden, that is it is only visible in the editing text and will not be rendered in the output HTML. **It is recommended to use the Section tools in BlogManager and avoid manual entries as described here and better management.**
`-+`	`-+++-` `\-+ not recognized`	**Create an aside** Dash and one or more + signs, final dash optional. Create an <aside> element. Remark: To close the aside before the end of the text open a new section using `-=` .

marker	example	description
-_	-____- \-_ not recognized	**Create footer** Any number of underline signs, final dash is optional. ***Remarks***: This marker should be avoided if the template already has a footer to host the category and tags information. **No more than one footer** should appear in one article (the whole content).

Headings

marker	example	description
The following markers are recognized if placed at the beginning of the line only.		
#SP	# Title ## minor title \# not recognized #not recognized # Title ======= Lorem ispum. render: **Title** Lorem ipsum.	**Create heading** (artbitrary) *(see also below)* Hash (#) followed by one space. Heading, converted to <h1>. Adding more hashes creates a lower heading, so for example three hashes create a <h3> heading. No more than six hashes are allowed, if more than six are met these are treated as a simple series of hashes and will not be interpreted. ***Remarks***: An immediately following line with two or more equal signs (==) is ignored, but can be useful for better readability of the text while editing.
Heading text == BLANK LINE *Heading text* -- BLANK LINE	Title ===== Minor title ---------- render: **Title** **Minor title**	**Create heading** (automatic) *(see also above)* Writing a line of text, followed by a line with almost two = or two – , followed by a blank line makes a heading. This is alternative to the # marker. = creates a <h2> (or <h3>) heading – creates a <h3> (or <h4>) heading This method do not allow to create <h1> headings. The heading is automatically demoted to one level if the text is inside a section.

Formatting, comments, and special elements

Except where otherwise stated, all the following markers are recognized everywhere they are placed in the text.

marker	example	description
The following markers are recognized everywhere in the text.		
`*`	`*text*` `* text recognized differently`	**Make bold text** Asterisk followed by a character (non space). Convert *text* to <b> (bold). The missing ending asterisk should be avoided.
`*.`	`*.text*` `*. text*` `not recognized`	**Make** *italic* **text** As above, but convert to <i> (italic).
`**`	`**text**` `* *text**` `recognized differently`	**Make** ***strong*** **text** As above, but convert to (stylized to be rendered as bold and italic).
`*!`	`*!text*` `* !text` `recognized differently`	**Make** *emphatic* **text** As above, but convert to (stylized to be rendered as italic).
`*- ... -*`	`*-text-*` `*- text-*` `not recognized`	**Make** ~~strike-through~~ **text** Dash followed by a character (non space).
`*^`	`alpha*^beta` `true*^{citation needed}` `mass*^{a+b}`	**Superscript** Text inside optional braces is put in a superscript. To insert a brace inside the braces escape it with a backslash. Note: Do not use superscript for foot notes, but put a number inside brackets (see Auto footnotes). Remarks: Inside a superscript block no other markers are recognized, to apply a style to a superscript put the style outside the block. Do not use this syntax for superscript inside math blocks.

marker	example	description
*_	alpha*_{beta gamma} term*_a term*_b	**Subscript** Text inside optional braces is put in a subscript. To insert a brace inside the braces escape it with a backslash. Remark: Inside a subscript block no other markers are recognized, to apply a style to a subscript put the style outside the block. Do not use this syntax for subscripts inside math blocks.
[:c:]		**Centered paragraph** Put at the beginning of the line creates a paragraph of type: <p class="bm_center_"> ***Remark***: The class **bm_center_** must be defined in bm_styles.css file.
*	* First item.SP Still belongs to the first item. * Second item.CR New line same item. BLANK LINE ^important blank line at the end! Render as: • First item. Still belongs to the first item. • Second item. New line same item.	**Creates a dotted (unordered) list** Asterisk followed by a space. Creates a new entry in a list: . The text that follows will be included into the entry, each new line will be deemed as part of the same line. A new line in the same entry (with a break) can be achieved by appending a full point (period) right before carriage return. Conversely, appending a space or a continuation mark _ won't break the line. A new asterisk+space closes the previous entry and creates a new one. A blank line ends the list. Spaces or tabulations in source text are ignored (but after a period at the end of the line) in building the list, so you can align your text without affecting the end result.

marker	example	description
`1.SP`	`1. Chlorine` `2. Bromine` `1. Oxygen` Render as: 1. Chlorine 2. Bromine 3. Oxygen `1. Iron based:` `* Steel` `* Cast iron` `2. Copper based:` `* Brass` `* Bronze` Render as: 1. Iron based • Steel • Cast iron 2. Copper based • Brass • Bronze	**Creates an ordered list** Any number, up to two digits, followed by a dot and a space. The list follows the same rules as for unordered lists (see above). It is possible to nest unordered lists inside an ordered list, see example 2.
`_CR`	`A line of text. _` `This line follows.` A line of text. This line follows. `Another one.` `This is a new line.` Another one. This is a new line.	**Force line continuation.** Normally **. ? ! :** followed by carriage return (CR) makes the line to break. Adding a space after them causes the line to continue. The underscore is a visual aid. *Remark*: If a blank line follows then a new paragraph is created.
`-.-`	`Example: -.-` `This is a new line.` Example: This is a new line. `Example b:SP` `This line follows.` Example b: This line follows.	**Force line break.** Forces to break the line and carry to a new line. Only **. : ? !** followed by carriage return insert line breaks.
`\`	`\*`	**Escape character** Useful when a marker should not be interpreted but used as it is.

marker	example	description
`\% ... %/` `\%> ... %/`	`\%disabled section%/` `\%>inline%/`	**Disabling blocks of text** Disable and re-enable mark parser. The block of text inside the two markers \% and %/ is copied as it is in the output buffer, the markers are not copied though. Current paragraph or list is closed before inserting in output the content inside these markers. If a > immediately follows the \% marker, then the current paragraph is not closed and the block is put in-line.
`{!! ... !!}`	`{!!comment!!}`	**Comment section** This is not parsed nor copied into the output buffer, it is simply ignored.
`[0) ... :]`	`[1)text:]` `style 1`	**Enumerated style** Render the text in a specific style, the style is numbered and provided by the calling program, if a style is not found the marker is ignored. This creates a entity. The `:]` sequence terminates the entity that enclose the style. In bm_styles.css file the styles are defined as `.bm_enumstyle_0` through `.bm_enumstyle_9`. If a comment follows the style, that comment is used as a description in the menu Format > styles
`[:_name_: ... :]`	`[:_red_:this:] color` rendered as: this color provided the following style is given: `.bm_n_red_{color:red}`	**Named style** Create a element with the class specified by *name* . The class should then be defined in bm_styles.css file or <style> section in template files. *name* must begin and end with an underscore.

marker	example	description
`[:c{` ` ...` `}--]` `[:c:{<:class:>` `...` `}--]` `[:c{{!!lang!!}` ` ...` `}--]`	`[:c{` `here some code` `}--]` `[:c{{!!lang!!}` `if(a==b) { ... }` `}--]` `[:c{<:backblack:>{!!` `lanB!!}` `For i = 0 To 9` `  Print "Hello"` `Next i` `}--]`	**Code** Render the code in an element <code>. Markers inside this block are not recognized with the exception of `[* *]` to highlight, `[! !]` to mark an error and `[? ?]` to mark a brakpoint. The following markers can also be used right after the opening mark: `<:classname:>` optional, specify a CSS class in classname for this element <code>. Name should be smaller than 30 characters. `{!!lang!!}` optional, interpret style syntax like C/C++/Java/javascript languages to color keywords, #preprocessors, quoted strings, and custom keywords. `{!!lanP!!}` as for `lang` but only uses custom keywords. `{!!lanB!!}` for style syntax like Basic language. `{!!mark!!}` for markup HTML/XML (recognizes only `'<'` and `'>'` as mark delimiters. Remark: No spaces are allowed after the opening marker to specify classname and language. If used both classname must appear first.
`[:d.`*`stylename`*`{` *`content`* `}--]`		**Named style block** Create a block (`div` entity) with the style defined by *stylename*. Styles are defined in bm_styles.css (see named styles). Style blocks can be nested.
`__`CR		**Bordered table** Two underscores followed by carriage return. Instruct to render the following table with borders. After this marker it is expected a table. This marker is recognized only if it is at the beginning of a line before the table.

marker	example	description
TAB	TABThis is an indent TABparagraph. TABTABThe beginning TABof this line is TABindented further. It is rendered as: This is an indent paragraph. The beginning of this line is indented further.	**Indent text** Creates a paragraph (element <p>) that is indented proportionally by the number of tabulations: 8ch per tabulation. Following lines that belong the same paragraph still are indented, even if in the text they are not. To make the text nicer to read, the same number of tabulations can be included at the beginning of the following lines, without affecting the output. Further tabulations add a element with left padding in ch proportional to the number of tabs (again, 8ch per tab). **Caution**: If TAB follows other characters (including space) from the beginning of the line, then it would be interpreted as a table! TABs are not recognized inside lists.
*> *>()	*>Simple quotation *>(caption) text *>("citation") text *>(http://url.tld "citation") text *>(caption "citation") text *>(caption https://url.tld "citation" more caption) text *>(caption https://url.tld "citation" more caption, and http://w3c.org "more citation") text.	**Quotation** *> marker must be at the beginning of the line. (caption) optional, if used must follow *> . Within parentheses other parentheses must be escaped using backslash (\ ), example: \(and \) the same apply for double quotes ("). Within parentheses, double quotes delimits citations. If citation is preceded by http:// or https:// then the citation is made into a link. Within double quotes parentheses must not be escaped, but double quotes themselves muts. After parentheses the quoted text ensues. Within parentheses more citations are possible. The quotation block must follow a blank line (or be the very first character in text). To end the quotation block add a blank line.

Links and anchors

marker	example	description
`[:#anchorname:]`	`[:#summary:]`	**Create an anchor** Anchor in page that can be addressed through a link (see how to create a link, below). After the sequence `[:#` enter the name of the anchor, followed by `:]` . The name should be no more than 50 characters long. Tip: If you put an anchor right after a heading, this heading will have the **id** with the name of the anchor. If you put an anchor in other locations a new element is created. Note: Do not confuse with shortcut local link.
`NS[`*`<number>`*`]` *`[<number>]`*`:`	`Statement[4].` `[4]: Foot note about statement.`	**Automatic Footnotes** Automatic detection of note references and foot notes. A note reference is a numeric reference enclosed in brackets that must follow the text with no spaces (NS). The related foot note is detected when the same number is enclosed in brackets followed by colon. **Remark**: No checking is performed: you can enter a note reference to a foot note that does not exist, and vice versa.
`[:link:`*`address (title) text`*`:]`	`[:link: {{relativepath}}doc.pdf (a pdf document) read document:]`	**General link.** This creates a link that is good for all purposes but anchors and intra-page links. In many cases it is easier to use the other forms described below, but for certain links that do not start by http:// or https:// the use of `[:link:` is necessary.

marker	example	description
`[:#: shortcut link :]` **Do not confuse with:** `[:#anchorname:]`	`### Topic one` `... some text ...` `For more info go to:` `[:#: Topic one :]`	**Shortcut link** To a local anchor, example a heading. In place of typing the full expression: `[:link:#<anchorname> (<title>) <text>:]` The anchor ID is taken from the text, that should match the anchor's name (example, the heading text). An optional description can be included this way: `[:#: (description) Anchor's name :]`

marker	example	description
Example of autodetected link: `https://mysite.tld/page#sec` `is rendered as:` https://mysite.tld/page **Example of full fledged link:** `[:@:linkaddress (title) text of the link:]` Example: `[:@:www.wikipedia.org (open wikipedia) go to wikipedia:]` Translates into: `<a href="www.wikipedia.org" title="open wikipedia">go to wikipedia</a>` `rendered as:` gto wikipedia Example: `[:@:mailto:info@accidentalscience.com email to accidentalscience:]` Translates into: `<a href="mailto:info@accidentalscience.com">email to accidentalscience</a>` `rendered as:` email to accidentalscience **Example of link to an anchor in the same page, supposing the anchor is named 'chapter1":** `[:@:#chapter1 go to chapter 1:]` Translates into: `<a href="#chapter1">go to chapter 1</a>` `rendered as:` go to chapter 1	**Create a link** Any chunk of text that begins with http:// or https:// is automatically converted into a link (**autolink**). The rendered text has the part the follow ? or # removed. The <a> entity created has the class attribute set to `.bm_a_` . To create a link with the address hidden and an optional title, or links for other protocols than http, start the sequence with `[:@:` immediately followed (with no spaces) by the address. Then, separated by a space or a new line, it should follow the text to display. Before this text an optional title can be added wrapped in parentheses. The link **must** be closed with the sequence `:]` . If the URL starts by http:// or https:// then it is possible to avoid `@:` so the following is valid: `[:https://wikipedia.org (Encyclopedia) Wikipedia:]` *Notes*. The title and text should not exceed 128 characters or an error occurs. To link a local anchor use the same name, example: `[:@:#anchorname go to anchorname:]` Within the title more parentheses can be inserted provided that each open parenthesis matches its closing one. Example, the following will produce an error: `(open (wikipedia)` While the following is legal: `(open (in) wikipedia)` If no title is provided then the text should not contain any parenthesis. You can provide an empty title as: `[:@:address () text:]` `[:http` or `[:https` and autolinks create links to a new browsing context *(since 1.0.1.21)*.

marker	example	description
`[:@<index> (title) text of link:]`	`[:@9 (open article 9) see article 9:]`	**Create link to a page** This creates a link to a page, the <*index*> should be the index of the page as listed in the program. Through the menu Link it is easier to enter the link of a page.

Remarks: Links created with `[:@:` and `[:@` have no class attribute.

Special characters

marker	example	description
The following sequences of characters are converted into equivalent HTML entities:		
`(c)`	You can also enter ©	Converts into copyright (©) ©
`(R)`	You can also enter ®	Converts into registered (®) ®
`(t)`	You can also enter ™	Converts into trademark (™) ™
€		Converts in €
£		Converts in £
&		Converts in & Use escape to leave it unmodified, example: \& do not convert the output.
<		Converts in < Use escape to leave it unmodified, example: \< do not convert the output.
>		Converts in > Use escape to leave it unmodified, example: \> do not convert the output.
␣<-␣		(space before and after) Converts in ← (←)
␣->␣		(space before and after) Converts in → (→)
␣|^␣		(space before and after) Converts in ↑ (↑)
␣|v␣		(space before and after) Converts in ↓ (↓)
␣<->␣		(space before and after) Converts in ↔ (↔)
␣<=␣		(space before and after) Converts in ≤ (≤)
␣>=␣		(space before and after) Converts in ≥ (≥)

marker	example	description
`----`		(every four consecutive dahses) Converts in —— **Remark**: After the first four dashes, the following are detected only in groups of four.
`CR` `---CR`		Isolated three dashes: convert to <hr> (horizontal rule). **Caution**: If the three dashes are not preceded by a carriage return (CR) then it would be interpreted as heading <h3>.
`1/2.`		Converts in ½ (½)
`1/4.`		Converts in ¼ (¼)
`3/4.`		Converts in ¾ (¾)
`(!)`		Renders as alert symbol ⚠
`\`	`Examples:` `\<  \>  \&  \{{` `\*  \#  \-  \[` `\:  \\` `[c)` would render: `<code>` `\[c)` would render: `[c)`	**Escape** Escape sequence is used to skip the interpretation of the following characters: < > & * # - [: and the couple of braces {{ . Example: the following sequence would be interpreted as code initiator: [c) however it would not by escaping the bracket: \[c)
`1st col 1st rowTAB2nd col 1st row` `.` `1st col 2nd rowTAB2nd col 2nd row` `BLANKLINE`		**Table** Create a table, where each row separated by a line with just a dot becomes a row of a table, and each column is separated by one or more TABs. The table ends with a blank line. (see Tables)

Grids (Table grid)

Table grids are similar to tables but they have a flexible structure that the browser can rearrange, adapting to every device.
To create a table grid just do the same as for tables but right before the table begins add the following command: **`$\tablegrid:`** after the colon you can enter CSS instructions to describe the grid. You can also add an ID that is assigned to the `<div>` element created.
The command ends when the line ends, so you cannot split CSS instructions over multiple lines of text.
Remarks: You don't have to provide the CSS display property as it is already set with the command. The optional ID can be used to set an ID name for the root element of the grid that you can use to adderess with CSS selectors and scripts. The optional ID name must contain a-z, A-Z - and _ characters only, with no spaces, and must not exceed 20 characters.

Example:

```
$\tablegrid:grid-template-columns:1fr 1fr;width:100%;ID:mygrid
```

Tables

To create a table just use one or more TAB characters to separate the columns. Each row must be separated by one line with just a dot (or dash or equal sign, see below), while the end of the table is marked by a blank line. Trailing TABs are ignored.
You may notice that even a paragraph with tabulations is interpreted as a table, which is useful to create tabulated text. This is important because within tables some markers are not recognized, in particular headings, ordered lists (but unordered are recognized), and nested tables.
To make the table more readable you can enter a colon or a pipe (' | ') character at the beginning of each column. This first character is not rendered to the output. Use a colon to indicate vertical alignment to the middle. Using pipes or colons is particularly helpful with empty cells.

Remark: Tables cannot be nested. It is recommended to check the result using the preview as it could be complex to make tables with just plain text.

Normally tables are rendered without borders. However prepending two underscores __ followed by a carriage return just before the table it will render the table styled *bm_bordertable_* which by default defines a table with borders.

Example:

```
__
First row         here some text for the first row,
                  a second line do not break the row and the text is
                  inserted into the same column determined by the TABs.
.
Second row        This text fall into the second column of the second row.
```

Each single TAB marks the separation between one column from the other. The TAB (or more consecutive TABs) need to be followed by some text to be considered a column separator, while trailing TABs are ignored. So in the text you can align visually the columns without affecting the result.

The following are equivalent and produce the same two column, one row, table:

```
First row   here some text for the first row.
```

```
First row         here some text for the first row,
```

To start with an empty column enter a pipe (|) followed by one (or more) TABs:

```
|TAB TAB here some text for the first row,
```

Implicit table

If the line begins with just TAB(s) no table will be interpreted, but an indented paragraph. However if more TABs are found in the same line then a table is implicitly interpreted.
Note that further lines lines after a table is initiated are allowed to begin just with a TAB and be interpreted as belonging to the same row but to the following column.
Text that follows in a new line with any blank line or a line that begins by a dot (.), dash (-) or equal (=) is gathered into the same row at the respective column defined by the TAB (remember consecutive TABs are read as just one TAB, so to count columns just look at the white space):

```
First row         here some text for the first row,
                  further text here still first row.
```

Result:

First row	Here some text for the first row, further text here still first row.

```
First row         here some text for the first row.
This goes into the first column because there is no leading TAB.
```

Result:

First row This goes into the first column because there is no leading TAB.	Here some text for the first row.

As in normal paragraphs, a period, colon, exclamation point or question mark followed by a carriage return breaks the line (inside the same cell). To force text remains in the same line (within the same cell) append a space and underscore going to the new line or before the TAB, example:

```
First row.              Long text in this cell. _
New line same cell.     But text follows in the same line.
```

Result:

First row. New line same cell.	Long text in this cell. But text follows in the same line.

Column order depends by one or more consecutive TAB(s) that are considered a single "white space":

```
First row, first col.  TAB   TAB  Second column.
 TAB This will go into the second column.
```

Is the same as:

```
First row, first col.  TAB   TAB  Second column.
  TAB   TAB   TAB   TAB This will go into the second column.
```

Result:

First row, first col.	Second column. This will go into the second column.

To create a second row just add a line with a dot in between:

```
First row, fist col.    First row, second col.
.
Second row, first col.  Second row, second col.
```

Result:

First row, first col.	First row, second col.
Second row, first col.	Second row, second col.

If you want to display a dot in a cell, just prepend a space so to make the dot not to happen at the beginning of the line or after a TAB (in other words at the beginning of the cell).

Explicit tables, Empty cells, Vertical alignment

An explicit table is detected when one or more TABs are met along a line. Explicit tables are those that have a colon (' : ') or pipe (' | ') character as a delimiter of a column. This first character is not rendered to the output. This makes it easier to create tables with empty cells.
An empty cell is detected when just a colon or a pipe is followed by one or more TAB characters, as shown below.
Since the first colon or pipe is never shown in output, to show this character type it a second time or prepend a space (enhnanced in yellow in the example below).

```
First       |                Third (first row)
.....       ............    ................................
|           |                | Third (second row), with pipe visible
.
Some rows   |                :This cell has the text aligned to the middle
in this     |                :
cell.       |                :
.
:           :                :| <- this pipe will be shown
```

First		Third (first row)
		| Third (second row), with pipe visible
Some rows in this cell.		This cell has the text aligned in the middle
		| ← this pipe will be shown

Remarks:

The pipe and colon characters used to mark the start of a cell are optional (see Implicit Table).
You like you can omit | or : in further lines of the same row.

Cell dimension

Cell width is automatically calculated using the number of characters in each cell. If a cell contains just an image, the size is set to a minimum of 4 characters.

To avoid possible disproportions you can enter more dots per each column below the first row. Dots reserve the space for the cells. Equal and dash signs have the same effect.
See examples in the following section.

Make table headings

To create a table heading add a series of dashes, equal signs below the row of text:

```
Heading first col       Heading second col
----------------        -----------------
First col and row       Second col first row
```

Result:

Heading first col	*Heading second col*
First col and row	Second col first row

Dashes (-) creates a simple thin line below the heading (see example above), equal signs (=) create a full featured table heading:

Heading first col	*Heading second col*
First col and row	Second col first row

Space reserver

As mentioned in Cell dimension dots, dashes and equals below the first row reserve space for the respective column. Dots create an invisible heading, that is useful to set the size (width) of the column to adapt to longer texts in following rows or in the case the cell doesn't have a meaningful size such as when is filled with an image. Example:

```
short                   Heading second col
```

```
.................       ==============================
Longer text here        Second col first row
```

short	*Heading second col*
Longer text here	Second col first row

Without the dots the result would have been:

short	*Heading second col*
Longer text	Second col first row

Remarks:

If the type of heading used in the first column is dash, this will command all the other headings of the same row that will be dash as well.

To end a table it is important to add a blank line.

Headings can be inserted even below from the first row, and there is no need to fill text above the heading lines:

```
|                       Heading second col
.................       ==============================
Longer text here        Second col first row _
                        with text on a new line within the same cell
.
*Another* row here
.
                        New heading in second col
                        ======================
Text in first col       Text in second col
```

Results:

	Heading second col
Longer text here	Second col first row with text on a new line within the same cell
Another row here	
	New heading in second col
Text in first col	Text in second col

Bordered tables

The marker __ (two underscores at the beginning of a new line and followed by a carriage return) direct the parser to interpret the following table with borders.

Example:

```
__
|                       Heading second col
.................       ============================
Longer text here        Second col first row _
                        with text on a new line within the same cell
.
                        New heading in second col
                        =======================
Text in first col       Text in second col
```

Result:

	Heading second col
Longer text here	Second col first row with text on a new line within the same cell
	New heading in second col
Text in first col	Text in second col

Size of the columns

Columns are automatically sized by the ratio of the number of characters used into each column over the total number of characters used in a single line, taking in account only the first row.
Therefore if the first row of one column contains text that is shorter than the one that appears in the rows below, that column could become too narrow.
Using an invisible heading made of dots is useful to force the size of the columns.
This works only for headers on the first row of the table.

Allowed markers within tables

Within tables most but not all markers are allowed: asterisks for emphasizing text (bold, italic, strike-through), unordered lists (asterisk followed by a space), special sequences that convert in meta characters such as +- to convert into ± , styles [x) :] and [:c{ code }--] sections, links and pictures.

Sections and headings are not allowed, if a hash sign (#) is met it is returned as is in the output and it will not be interpreted as a marker.

Nested tables are not allowed.

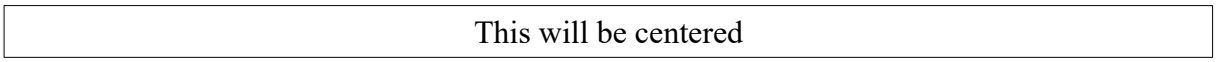
Centered text

Centered text can be made by entering `[:c:]` from the beginning of the line.

Example:

```
[:c:] This will be centered
```

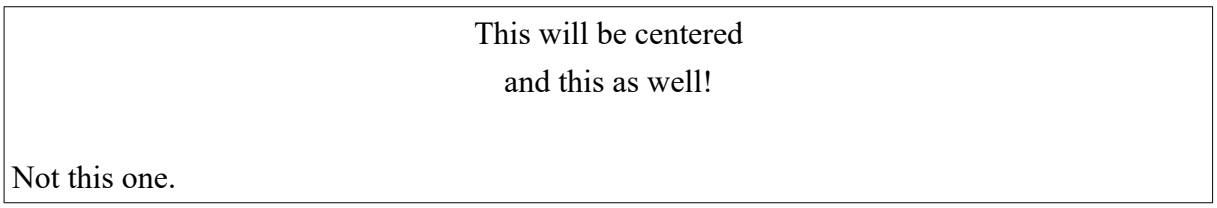
Result:



The paragraph remains centered as long as no blank lines are found. The following will create a single centered paragraph.

```
[:c:]CR
SPSPSPSPSPSPSPSPSPSPSPSPSPSPSPSPSPThis will be centeredCR
SPSPSPSPSPSPSPSPSPSPSPSPSPSPSPSPSPand this as well!CR
CR
Not this one.
```

Result:



You can add spaces to make the text to appear centered in your text source as in the example above, as the spaces will be ignored.

Template Files

Templates are filled with *elements* into special marker placeholders. Those markers begin and end with double braces. Elements can be either content (such as the list of tags/keywords) or external *element templates* picked from a file (and filled themselves with the required markers).
Each template page is always filled with the *common markers*, and may have one or more `[:include:` *markers* for the inclusion of external files, for more information see the chapter Template Markers Summary.

Templates are divided in two groups: *page templates* and *element templates*.
Page templates provide a model of the intended page, while element templates provide a model for a snippet of HTML code that will be used in place of a particular marker, sometimes when that snippet represent a record, it could be used several times to populate a list of data of the same type such as the list of tags.
Element templates usually have the extension ".h" while page templates have extension ".html".

Entry page

When the site is generated the entry page (typically index.html or welcome.html) is populated with the most recent articles, as specified into the template for the entry page, for the number or articles specified into the field Max number of most recent articles to list in **Site Settings**.
However if more articles are available, a new secondary index page is created and populated with the remaining number of articles per page, and so on. Those secondary index pages are enumerated from 1, example: index1.html, index2.html, etc.
The `{{URL_page_before}}` and `{{URL_page_next}}` markers are thus populated with the appropriate links that connect all the pages together.

Remarks.
- Each single element of an article that is included into the index is given by the *recentrecord.h* file.
- Hidden articles or pages are not included.
- The max number of most recent articles is also used to limit the number of articles to insert into a post page. In that case no more than this number of articles are listed.

Page as Index
If a Page has been marked as Index this will take over as entry page. A specific template is used to populate this page. Unlike the indexes mentioned in the previous paragraph a Page as Index is not filled with the most recent articles. However a full list of pages, articles A and B can be filled if the appropriate marker and selectors are found in the template. For more information: Page as Index.

Images in templates

When the template is loaded is initially scanned to search for all <img> tags and their 'src' attribute to collect the required images, those images should be located in the template path or a sub-folder if that sub path is specified in the src of the image. The file is then copied from the template path to the destination path into the same sub folder (creating it if not existent) as specified in the src of the image.

Example, given the following <img> element into a template:

```
<img src="myimages/image1.png" alt="image no.1" >
```

and assuming the templates path is in *C:\mytemplates*, the file *image1.png* should be located into the sub folder *myimages* , thus the full path for image1 would be: *C:\mytemplates\myimages\image1.png*.
Assuming the destination path for the generation of the site is in *C:\myfiles* and the site name is "my first site", the file would be copied into: *C:\myfiles\my first site\myimages*.
In the case the sub folder is not existent the program will create it.

Caution: Path to images must be relative as in the example above, if the path is a full link beginning by http:// or anyway contains the colon sign, or if it is prefixed with a double-brace marker (i.e. `{{relativepath}}`), or if it begins by `data:image` the <img> element is not processed.

Special Images

Images defined in **Site Settings**, that is: page banner, logo, and the icon indicating an article has a video (and other future extensions), are copied into the special sub folder *cmimgs* which is created into the destination path. Assuming then the destination path is *C:\myfiles* and the site name "my first site", the path for those special images will be: *C:\myfiles\my first site\cmimgs* .
As well, the images specified in the rootcompanionfiles.txt will be placed into the same special common image folder (see *Companion files and special folders cmimgs and docs* for details).

Companion files and special folders *cmimgs* and *docs*

The file ***rootcompanionfiles.txt*** located in the templates path can provide a list of files to be copied into the destination site path. Each item must be identified by the type of file followed by colon and the path and filename of the file. If the specified path does not have a drive separator (example: C:) then it is considered relative to the **Templates path**.
Destination path is implicit for the file type, but it can optionally be specified appending an asterisk (*) and the destination path, after the path and filename.
The destination path is always referred to the choosen rood directory where the site is generated.
Example:

```
file: C:\website\resources\webfont\fnt.woff * webfont
file: ..\tempfonts\temfnt.woff * webfont
```

Note that the spaces before and after the path-filename and path are ignored. Supposing the root directory where the site is generated is in *C:\myfiles* (destination path), and the site name "my first site", the line above will direct the program to search for the file fnt.woff in *C:\website\resources\webfont* and it will copy the file into *C:\myfiles\my first site\webfont* .
The destination path is created if not existent.

Four types of files are defined:

img:	provides images that will be copied into the common image folder *cmimgs* if not otherwise specified with an optional destination path separated with an asterisk;
pdf:	are document files that will be copied into a special sub folder named *docs* (if the folder do not exists it is created), if not otherwise specified;
file:	are other files that will be copied into the destination root, if not otherwise specified;
page:	are HTML pages that will be copied into the destination root, if not otherwise specified.

In addition the following are defined:

page-unlist:	as for page, but it is not listed in sitemap.
pdf-unlist:	as for pdf, but it is not listed in sitemap.
list:	This is a link to a resource already available. This file is not copied but its link is added to sitemap.

Syntax of **list:** *source_pathfilename_on_disc * URL_path*

URL_path must be the path referred to the local website, without the the filename (which is taken from the *source_pathfilename_on_disc*).

source_pathfilename_on_disc Full path and file name of the resource to list, as if it were located into the local disc as for other files.

Example:

```
list: C:\mypath\example.html * myspecial/path/for-example
```

myspecial/path/for-example is meant to be found from the root of the website, so this resource must be uploaded manually or through different tools.

Unlike **pdf-unlist:** and **page-unlist:** the files identified as **pdf:** or **page:** will be included into the sitemap.xml file, and sitemap.html.

The files identified as **page:** and **page-unlist:** are scanned for the images contained in `<img>` elements, copying the file identified by the *src* attribute as specified in the above section Images in templates.

In addition pages can contain any common markers (see *Common Markers* sub-section below) that will be replaced with the proper data before copying the page into the destination root directory.

Page Templates

filename	description
templatepost.**html** templateArtB.**html**	Template to build a post page for an article class A and B respectively. Allowed markers: {{site_name}} {{site_tagline}} {{site_description}} {{site_taglist}} *(see also tagrecord.h)* {{article_content}} {{article_related_content}} {{article_link}} {{article_fullLink}} {{article_title}} {{article_date}} {{article_author}} {{article_abstract}} {{article_posterURL}} {{article_poster_depiction}} {{article_poster}} {{article_video}} {{article_related}} *(see also relatedrecord.h)* {{articles_recent}} *(see also relatedrecord.h)* {{article_tags}} {{article_taglist}} {{article_category}} {{URL_article_category}} {{URLimgpath}} {{URLarticlespath}} {{URLsitepath}} {{relativepath}} {{entry_page}} {{entry_pageURL}} {{search_page}} {{search_pageURL}} {{root}} {{pagebanner}} {{logo}} {{categories}} *(see also categoryrecord.h)*
templatepage.**html**	Template to build a generic page not associated with articles (see Articles and Pages). The same makrers as for *templatepost.html* are allowed (using the same name "article"). However the following, are not allowed: {{article_abstract}} {{article_related}} {{article_recent}} Note that {{article_related_content}} is still allowed.

filename	description
templateindex.**html** see also: templatepageindex.html	Template for making the entry page (index or welcome.htm) and sub index pages. This is used for all indexes including the entry page if no special page index is selected, otherwise for the main entry page is used templatepageindex.html (see below). See also Page as Index. Allowed markers: {{site_name}} {{site_tagline}} {{site_description}} {{site_taglist}} *(see also tagrecord.h)* {{URLsitepath}} {{entry_page}} {{entry_pageURL}} {{search_page}} {{search_pageURL}} {{relativepath}} {{root}} {{pagebanner}} {{logo}} {{curdate}} {{URL_page_before}} {{page_before}} {{page_beforeSnippet}} {{URL_page_next}} {{page_next}} {{page_nextSnippet}} {{articles_recent}} *(see also recentrecord.h)* {{categories}} *(see also categoryrecord.h)* Index of pages: {%xtract:pages_index%}`INDEXBODY`{%/xtract:pages_index%} `INDEXBODY` may contain: {{article_link}} and {{article_title}} (see **Articles and Pages**).
templatesearch.**html**	Template for making the search page. Allowed markers: {{site_name}} {{site_tagline}} {{site_description}} {{site_taglist}} {{URLsitepath}} {{relativepath}} {{root}} {{pagebanner}} {{entry_page}} {{entry_pageURL}} {{search_page}} {{search_pageURL}} {{logo}} {{categories}} (see also categoryrecord.h)
templatesitemap.html sitemapHTMLrecord.h	Template for generating a human readable sitemap. Also requires sitemapHTMLrecord.h. Markers recognized: {{MapSearch}} page Search {{MapIndexes}} all Indexes {{MapArticles}} all Articles A {{MapArticlesB}} all Articles B {{MapPages}} all Pages {{MapOther}} all other files (from rootcompanionfiles.txt) {{MapURL}}, {{MapTitle}}, {{MapLasdModified}}

filename	description
bannerheader.html lowerbanner.html	Blocks of HTML code that is inserted into the heading portion and lower portion of templates for articles and pages.
notfound.html	This file is required. It is populated by BlogManager and it will be the landing page for all broken links.
templatepageindex.html	This file allows to create a template for the main Index entry page alternative to the first index generated. See also Page as Index.

Element templates

filename	description
postersnippet.**h** postersnippetvideo.**h**	Snippet of HTML to embed a poster image for the article. It may contain the markers: {{article_posterURL}} and {{article_poster_depiction}} (textual description of the image used in the *alt* attribute). postersnippet.h — used for articles' poster image postersnippetvideo.h — used as a video card (using the poster image) Remark: The resulting element will replace the marker {{article_poster}} into the snippet **videolink.h**.
videolink.**h**	Template for the inclusion of a video link related to an article. Enter the link into the specific box **Video**, and make sure the checkbox **Video snippet** is not checked. Allowed markers: {{article_videolink}} and {{article_poster}} (aricle poster uses the poster image as a video card). Remark: The resulting element is then included in place of the marker {{article_video}}.
videosnippet.**h**	Template for the inclusion of a video snippet of code for the article. The code must be provided by entering it into the specific box **Video**, and making sure the checkbox **Video snippet** is checked. Allowed markers: {{article_videosnippet}} Remark: The resulting element is then included in place of the marker {{article_video}}.
relatedrecord.**h**	This template is the backbone to build a whole block containing one link to an article that is related to the main article. This template is

filename	description
	thus embedded into the post article page, and repeated for each related article. This is also used to embed the most recent articles in a post page in lieu of the recentrecord.h template. Typically this block is embedded into an <aside id="related"> element that is located inside the post article's page. Allowed markers: {{article_link}} {{article_title}} {{article_abstract}} {{article_poster}} {{article_posterURL}} {{article_poster_depiction}} {{article_category}} The maximum number of related articles to list, and the maximum number of recent articles to list are specified in **Site Settings** .
recentrecord.h	Similar to relatedrecord.h but used to insert the most recent articles into the entry page (index or welcome.htm) and sub index pages. After this record has been properly filled it is placed into the page, and a new record is thus appended up to the maximum number of most recent articles to list into the entry page. Then, if more articles are available, the same procedure is performed generating a new enumerated sub index page such as index1.html, index2.html and so on. Allowed markers: {{article_link}} {{article_title}} {{article_abstract}} {{article_poster}} {{article_posterURL}} {{article_poster_depiction}} {{article_video}} {{article_hasvideoicon}} {{article_category}} {{URL_article_category}} The maximum number of recent articles to list is specified in **Site Settings** .
categoryrecord.h	Similar to the previous one, this record defines one entry for the list of categories that is placed in the entry page. The same record could be included in other pages, such as a look up page or as an aside in an article page. It can contain: {{categoryname}}, {{URLcategoryname}}. URLcategoryname is the same as categoryname but URL encoded. A javascript function call allows to sort out all the articles related to that category. It receives the category name by inserting this marker. The result of the function would be either a list generated on the

filename	description
	same page, or pointing to a new page with the results. For more information see JSON Generated Files.
tagrecord.**h**	Similar to categoryrecord, holds one entry for the list (or cloud) of tags. It can contain: {{tagname}} {{URLtagname}} . URLtagname is the same as tagname but URL encoded. A javascript function call allows to sort out the articles related to that tagname and receives the tag name by inserting this marker. The result of the function would be either a list generated on the same page, or pointing to a new page with the results. For more information see JSON Generated Files.
figure.**h**	Snippet template for figures (pictures). It take place where the \ %<figure>...</figure>%/ tag is located within the text. It contains: {{figcaption}} description of the picture {{HTMLfigcaption}} as above but HTML encoded {{imgsrc}} path and filename of the image {{altimg}} alternative depiction of the image {{heightwidth}} populated with the height and width of the image, if specified.
nextsnippet.**h**	Snippet of HTML code that may incorporate the {{page_next}} and {{URL_page_next}} markers. This snipped is loaded to take place of the marker {{page_nextSnippet}} if found in the page and if a next page exist.
beforesnippet.**h**	Snippet of HTML code that may incorporate the {{page_before}} and {{URL_page_before}} markers. This snipped is loaded to take place of the marker {{page_beforeSnippet}} if found in the page and if a before page exist.
sitemapHTMLrecord.**h**	Snippet of HTML code used to build a record in sitemap.html.

Other files

robotrules.txt	Specify rules for robot.txt file.
custkeywords.txt	List of custom keywords that should be recognized when interpreting text from code sections.

JSON Generated Files

In addition to the file generated using the above templates, a number of JSON files are generated as well.

filename	description
articlesdb.**json**	JSON object with the whole database of articles, ordered by article index. The articles included are only those that are not hidden. *Data:* title, tags (blob), category, modified, URL. The order of the entries in the array implicitly is the index of the article.
categoriesll.**json**	JSON object with an associative array listing all the categories with a linked list of the articles per category. Hidden articles are not included. *Data structure*: `{ "categories": [ "categoryname": [ index, index, ... ], "categoryname": [index, index, ... ], ... ] }` Index is the article's index as implicitly set in order in articledb.json, so given a category name the corresponding key gives the list of indexes to the articles within the articledb.json array.
tagsll.**json**	JSON object with associative array listing all the tags with the related article indexes. Hidden articles are not included. *Data structure*: `{ "tags": [ "tagname": [index, index,...], "tagname": [index, index...], ... ] }` Index is the article's index as implicitly set in order in articledb.json, so given a category name the corresponding key gives the list of indexes to the articles within the articledb.json array.

Template Markers Summary

Templates are used as a model to build the intended type of page (entry, search, post page) and markers are used as placeholders for the inclusion of various elements. Markers are surrounded by double braces or brace percent sign. Markers can happen more times in a template, all the instances will be replaced with the intended values/elements.

Include Marker

A special marker `{{include:...}}` allows the inclusion of external files. This let have a single file common to many templates, such as a common script or heading file.
The inclusion is performed before any process of replacement of the marker placeholders, thus the included files could themselves have the very same markers as if the file were part of the template itself. Include markers can happen more than one time in a template, even with the same file to include.

Common Markers

A set of common markers are checked out and replaced in every template, at the beginning of the process. Common markers can be used in any template, including many *element* templates.

marker	description
Common Most of the following markers are filled with the information entered in **Site Settings**.	
`{{site_tagline}}`	Placeholder for the tag line. This marker could be placed in the title or in meta elements or other convenient locations such as the header of the page. CAUTION: this value is converted with UTF8 named character entities for characters above ASCII, therefore it cannot be used inside attributes.
`{{site_description}}`	Provide the site description, typically used in meta tag.
`{{site_taglist}}`	Placeholder for the list of tags. Tags are used as tag keywords related to articles or the whole site, tags are comma delimited and this placeholder is filled with the list of `keywords` entered for the site in **Site Settings** .

{{site_name}}	Replaced with the name of the site, as entered in **Site Settings**. CAUTION: this value is converted with UTF8 named character entities for characters above ASCII, therefore it cannot be used inside attributes.
{{pagebanner}} {{pagebannermobile}}	The image used as page banner, the page banner is usually the image that appears on the top of the pages. Use a CSS media selector to hide the standard banner or the mobile banner. Source file are searched in templates directory. See also {{video_banner}}.
{{video_banner}}	Replaced with the main video snippet of code if that code has a <video class="bm_vbanner" > and video snippet is checked.
{{logo}}	The image used for the logo.
{{entry_page}}	The name given for the entry page or the "home", that is the page the server will serve by default. Typically this is named index.html or welcome.html
{{entry_pageURL}}	As above but it is URL encoded thus suitable to be used in links.
{{search_page}}	The name given for the search page, this typically is search.html but can be named as you like.
{{search_pageURL}}	As above but URL encoded thus suitable to be used in links.
{{root}}	This is replaced with a slash (/) if the page is not located in the root of the site but in a subfolder. It is simply removed if the slash is not applicable.
{{URLsitepath}}	Filled with the site URL as set in **Site Settings**. Remark: The URL will never have any trailing slash, even if specified in Settings.
{{relativepath}}	Replaced with ' ../ ' if the article is not located in root, for other pages or elements located in root, or if the article is located in root, marker will be simply removed.
{{fontpath}}	Use this marker to specify the base path of your web font. Example: Suppose your fonts are located in C:\webtools\webfonts and you entered this path into the related field in Site Settings. Then suppose you have a font in C:\webtools\webfonts\webkit so in the css file you will have: `URL('{{fontpath}}webkit\fnt.woff')` This entry will be created in the rootcompanionfiles.txt file as: `files: C:\webtools\webfonts\webkit\fnt.woff * webkit`

`{{categories}}`	Replaced with the list of categories, if no category is available the marker is just removed. The list of categories is built combining a series of records, one for each category, using the content of the categoryrecord.h file, and replacing its markers.
`{{commonimgpath}}`	Replaced with the path to common images (cmimgs).

Special Markers

These belong to the common marker group, but have some special purposes.

marker	description
`{{include:}}`	Replaced with the content of the file specified after the colon sign. Example: `{{include:myfile.html}}` will be replaced with the content of the file 'myfile.html' that must be found in the same folder for templates. If the file is not found an error occurs. The inclusion happen at the very beginning of the process of replacement, so after the inclusion even the included content is parsed and markers replaced accordingly.
`{{curdate}}`	Replaced with the current date when the site is generated (or regenerated).
`{{curdatetime}}`	Replaced with generation date and time: hh:mm:ss dd-mmm-yyyy
`{{curyear}}`	Replaced with generation year (full year).
`{{curtime}}`	Replaced with generation time.
`{{welcome_content}}`	Replaced with the content of the Entry page welcome section.
`{{page_link_<pageID>}}`	Where `<pageID>` is the ID of a page. To change type between *page* and *article* choose menu `Article/Page > Change type` . Example: `{{page_link_Terms-of-use}}` . Replaced with the link to the *page*. To get the ID of the page click on the title bar above the list of articles/pages.
`{{page_fullURL}}`	Filled with the full URL of the current page, whatever it is. *(from 1.0.121)*

Marker Functions

These markers converts the text within them and return the converted text in place of the marker itself. The text must be separated by almost one space from the marker identifier and delimiter (below indicated by SP). The text may be the value returned by any variable marker (markers that are enclosed in double braces). Example: `{%esc: {{article_title}} %}` if, let's say the title of the article is "Blacksmith's tools", the returned text would be: `Blacksmith\'s tools` .

`{%xtract:` Marker Function is a special marker used to extract embedded mini-templates. It can be used to delete sections or to include (and repeat if necessary) properly filled sections.

marker	description
`{%esc:`SP*text*SP`%}`	Function escape characters. Returns the given *text* escaped, useful in javascript. *Text* can be the returned value from any variable (markers with double braces like `{{article_author}}`. Remark: Only double quotation marks are escaped, single are not. So it is not recommended to include the returned text inside single quotation marks.
`{%fuenc:`SP*text*SP`%}`	Function full URL encoding. Returns *text* URL encoded. Example: `{%uenc: my site.com %}` returns: `my%20site.com` Remark: The function performs full encoding, including the characters that are part of the URL syntax (see also ***uenc***). This is useful for links that should be included as argument of a parameter of another URL.
`{%uenc:`SP*text*SP`%}`	Function URL encode. Returns *text* URI encoded. This function do not encode the characters that are part of the URL syntax, namely `? & # , ; '` (see also ***fuenc***). This is useful for links with components.
`{%xtract:`*identifier*`%}` `{%/xtract:`*identifier*`%}`	Extract the chunk of text within the two markers. Used to extract templates from the template page itself. *Identifier* identify the operation. Important: no spaces are allowed inside this marker. **Recognized identifiers** *pages_index* — extract a template for the list of links to *pages* (in contrast with *articles*). *pages_menu* — extract the menu template that hosts the page_index, or it is removed altogether if

marker	description	
		there are no pages to add to the index.
	videobanner	template for video banner, which is an alternative banner that may host a video. The video snippet of code can be made by clicking Create... button and checking put as banner, be sure the Video snippet is checked as well. The snippet of code with `class="bm_vbanner"` is inserted into the main video text box. This will take place where the variable `{{video_banner}}` is found in the template. If no snipped of code marked as video banner, then `{%xtract:videobanner%}` and its content is removed from the template.
	banner	template for the page banner. It is alternative to videobanner. If no video as banner is specified then the content inside this marker is extracted and used, the variables `{{pagebanner}}` and `{{pagebannermobile}}` would be filled if used inside this block.
	author	extract a block and populate with author or authors if more than one. This is useful for <meta> , and everywhere authors should be listed in separate tags/elements such as in atoms and rich snippets. Example: `{%xtract:author%}` `<meta name="author" content="` `{{article_author}}{{article_coauthor}}">` `{%/xtract:author%}` Note: Only one of the two author/coauthor markers is filled: first record will be filled with author, following records with coauthor. The other element is automatically removed.
	sections_index	Delimits a block of HTML code that can be used as template to create an index of sections. Inside this block is expected to find a block *sections_index_item* (see below).
	sections_index_item	Delimits an embedded template to create an

marker	description	
{%xtract:*identifier*%} {%/xtract:*identifier*%} *(continue)*	*welcome_section*	template for welcome content. This is the content entered choosing Edit Entry page. It will go only on the first index.html (or welcome.html) page. In all other pages this marker and its content is deleted. The template inside this marker should contain {{welcome_content}} marker variable.
	welcome_before	Used to wrap an embedded template that is deleted if the welcome option **after** is selected. This is always deleted in pages and articles. See remark below.
	welcome_after	Used to wrap an embedded template that is deleted if the welcome option **before** is selected. This is always deleted in pages and articles. **Remark**. Other marker functions can be contained by the *welcome_after* and *welcome_before* markers.
	artfooter *artheader*	You can use these two markers to wrap the header and footer of a Page or Article, that can be removed if for that document options Hide header and/or Hide footer are checked. Document's options are available through menu Article/Page > Options
	hide_production *hide_test*	This can be used to wrap portions of HTML in your templates that need to be hid either in production (when in preview or test build) or in test (when site is built).
	related	Delimits a region of code that is meant to host the {{article_related_content}} marker. (From version 1.25)

Remarks: {%xtract: markers can contain other nested selective-extractive markers, as well as other marker functions (those that start by {%) and variables (those that start by {{).

Welcome Before and After selectors can be used to create two different types of embedded templates that goes before or after the menu (or whatever you like in your template). So while editing it is possible to change where the welcome content goes.

Markers specific for Articles, Pages and elements

marker	description
Specific These markers are specific for some templates, please also refer to the Template Files chapter.	
Specific for artciles/posts/pages for templatepost.html, templatepage.html templatepageindex.html, templateArtB.html . **Common markers apply, in addition the following specific markers are recognized:**	
{{article_content}}	Replaced with the content of the article. The content is converted from the text marker mode into HTML, links to other articles and pictures filled properly, and the required picture files copied from the source of images, specified in the Path to repository of images in **Site Settings**, into the intended destination sub folder (or in root if not specified in **Site Settings** through the checkbox Copy Images and textbox Server side sub-folder for articles' images).
{{article_link}}	Properly filled with the link to an article (URL encoded). Use {{article_linkURI}} for fully encoded URI suitable to be used as argument in a URL.
{{article_linkURI}}	Similar to {{article_link}} but fully encoded (that is also ? # are encoded), suitable to be used as an argument in a URL.
{{article_fullLink}}	Properly filled with the absolute link to an article (that is the link comprising the site URL), useful for permalinks.
{{article_title}}	Filled with the title of the article. CAUTION: this value is converted with UTF8 named character entities for characters above ASCII, therefore it cannot be used inside attributes.
{{article_date}} {{article_time}}	Filled with the last modified date and time of the Article or Page.
{{created_date}} {{created_time}}	Filled with the creation date and time of the Article or Page.

marker	description
{{article_author}} {{article_s_author}} {{article_coauthor}} {{article_s_coauthor}}	Filled with the author name as specified in **Site Settings** or into the field **Author(s)** of the article. Multiple authors can be entered separated with commas. Further authors will be filled in {{article_coauthor}} if present, otherwise the same {{article_author}} marker is used. CAUTION: this value is converted with UTF8 named character entities for characters above ASCII, therefore it cannot be used inside attributes. Use {{article_s_author}} for non UTF8 strings, also embrace it as {%esc: {{article_s_authos}} %} to escape double quotes where required.
{{article_abstract}}	Filled with the abstract of the article. The abstract is converted from the text marker mode to HTML. Links, images and other elements besides text should not be used into the abstract. See also {{article_description}}. Not allowed in *templatepage.html*.
{{article_description}}	Filled with the first 250 characters of the abstract. The string is stripped of any CR or LF and multiple spaces, and escaped for apostrophes and double quotation marks.
{{article_related_content}}	This is filled with the related section included into the article or page text (through menu Insert > Related text). Usually this marker is inserted into the extract section *related* (see Marker Functions above). Available from version 1.25.
{{copyright}} {{s_copyright}}	Filled with the copyright name entered in **Site Settings**. If the field is empty the first author is used in place. CAUTION: {{copyright}} is converted to UTF8, use {{s_copyright}} to include into quoted text, also consider using {%esc: {{s_copyright}} %} to escape double quotes. Example: `<script>` `  copyowner = "{%esc: {{s_copyright}} %}";` `</script>`

marker	description
{{article_list}}	Filled with a list of all articles of the current article class. The item is replaced with a block as in the following example: *article-title* <li class="bm_curitem"> *article-title* ... This is a default placeholder in the case the extraction marker {%xtract:artlistitem%} is missing into the block {%xtract:artlistA%} or {%xtract:artlistB%}

marker	description
{{article_poster}}	Replaced with the element postersnippet.h (see {{article_posterURL}} and {{article_poster_depiction}}) If Video snippet is not checked and the Main video edit box is not empty, this marker would be replaced by postervideosnippet.h and the poster image would be used as the image for the link to the video (the Main video edit box is expected to contain a valid URL to a video in this case).
{{article_posterURL}} {{article_poster_depiction}}	Replaced with the URL to the poster image and its depiction (alternative text), respectively. These values can be entered by clicking the Select image to insert... and checking poster.
{{article_video}}	Replaced with the element videolink.h or videosnippet.h in the case the Video snippet checkbox is checked **and** the code do not begins by <video class="bm_vbanner" otherwise it is meant as video banner and not the main video of the article. To create the snippet of code you can click the button Create... otherwise you can paste the code, i.e. to embed a YouTube player.
{{article_related}}	Replaced with the element(s) relatedrecord.h, or removed if no related articles are found. To seek for related articles the tags match criteria is used: the more tags that match the current article, the more likely the article is listed on top of the list, up to the limit set in Max related articles in **Site Settings**. Not allowed in *templatepage.html*.
{{articles_recent}}	Replaced with the element(s) relatedrecord.h with the most recent articles. The limit is set in Max recent articles in **Site Settings**. Not allowed in *templatepage.html*.
{{article_tags}}	Replaced with the element(s) tagrecord.h, or removed if the article has no tags. One or more record elements are collected and properly filled to create the list of tags that will take place in lieu of this marker placeholder.
{{article_taglist}}	Filled with the list of tags, as a string of keywords comma delimited.
{{article_category}}	Filled with the name of the category of the article. CAUTION: this value is converted with UTF8 named character entities for characters above ASCII, therefore it cannot be used inside attributes.
{{URL_article_category}}	Filled with the URL encoded name of the category.

marker	description
{{URLimgpath}}	Filled with the path to the images for articles (with appended slash if required).
{{URLarticlespath}}	Filled with the path to the articles (with appended slash if requred).

marker	description
Specific for templateindex.html (template for entry page, and sub indexes) Common markers apply, in addition the following specific markers are recognized:	
{{articles_recent}}	Replaced with the element(s) recentrecord.h properly filled, if no recent articles are found the marker is removed. The articles are listed following the date order, the most recently modified is put first. The number of articles listed is specified at the Max number of recent articles in **Site Settings**. If more articles than the set limit are available, more sub index pages are generated and enumerated.
{{URL_page_before}} {{URL_page_next}}	Replaced with the URL to the previous or next index page respectively.
{{page_before}} {{page_next}}	Replaced with the number of page before or next, respectively.
{{page_beforeSnippet}} {{page_nextSnippet}}	Replaced with a snippet of HTML code loaded from the file ***beforesnippet.h*** and ***nextsnippet.h*** respectively. If these snippet do exist they can contain the above markers related to page navigation. If no page before or no page next, the relative snippet is not loaded and the marker deleted.
{%xtract:pages_index%} {%/xtract:pages_index%}	These markers delimits a region of text that is used as a template for building the index of pages (not articles and not special pages listed in *rootcompanionfiles.txt*). Example: `{%xtract:pages_index%}<a class="menu_index_pages" href="{{article_link}}">{{article_title}}</a> {%/xtract:pages_index%}`
{{article_link}} {{article_title}}	Only allowed inside the delimited block for the pages index (see above). Respectively, the link to a page and its title.
{{article_list}}	Only allowed inside the delimited block selector {%xtract:artlistA%} and {%xtract:artlistB%} . Filled with the list of articles class A and B respectively.

marker	description
There are no specific markers for templatesearch.html So, only common markers apply.	

marker	description
Specific for poster elements postersnippet.h and postersnippetvideo.h Common markers apply, in addition the following specific markers are recognized:	
{{article_posterURL}}	Filled with the URL to the poster image file selected for the article, the poster image file is also copied into the article's images folder (whatever mode is selected in **Site Settings**). Remark: This element could be used either for replacing {{article_poster}} or {{article_video}} when the content of the Main video edit box is a link and not a snipped code and Video snippet is not checked.
{{article_poster_depiction}}	Filled with the depiction (the alternative description or *alt* attribute in the *img* tag) for the poster entered while selecting the image for the article's poster.

marker	description
Specific for the element videolink.h Common markers apply, in addition the following specific markers are recognized:	
{{article_videolink}}	Filled with the link to the video associated with the article and specified in the text box Video and making sure the checkbox Video snipped remains unchecked. Remark: the value is inserted as is from the value entered in the text box.
{{article_poster}}	Filled with the element postersnippetvideo.h if a poster image is specified for the article.

marker	description
Specific for the element videosnippet.h **Common markers apply, in addition the following specific markers are recognized:**	
{{article_videosnippet}}	Filled with the code (typically the video player) associated with the article and specified in the text box Main video and making sure the checkbox Video snipped is checked. Also the code should not begins by <video class="bm_vbanner_" but it could begin by <video class="bm_mainvideo_" . This element takes the place of the {{article_video}} placeholder.

marker	description
Specific for the element relatedrecord.h **Common markers apply, in addition the following specific markers are recognized:**	
{{article_link}}	Filled with the link to the article (URL encoded).
{{article_title}}	Filled with the article's title. CAUTION: this value is converted with UTF8 named character entities for characters above ASCII, therfore it cannot be used inside attributes.
{{article_abstract}}	Filled with the abstract of the article, the abstract is converted from text marker mode to HTML. Links, images and other elements such as tables should not be used in abstract.
{{article_poster}}	Replaced with the element postersnippet.h
{{article_posterURL}}	Filled with the image poster URL. Remark: This is redundant if {{article_poster}} is used.
{{article_poster_depiction}}	Filled with the depiction of the image (the *alt* attribute of the *img* tag). Remark: This is redundant if {{article_poster}} is used.
{{article_category}}	Filled with the category name associated with the article. CAUTION: this value is converted with UTF8 named character entities for characters above ASCII, therefore it cannot be used inside attributes.
{{URL_article_category}}	As above but URL encoded so it can be appended to a link, such as the one for search.html
{{article_date}} {{article_time}}	Respectively, last modified date and time of the Article.
{{created_date}} {{created_time}}	Respectively, creation date and time of the Article.

marker	description
{{article_author}} {{article_coauthor}} {{article_s_author}} {{article_s_coauthor}}	Author name, as specified in **Site Settings** if no Author is specified in the article's attributes on the left pane.
{{article_tags}}	Filled with the tag records (see tagrecord.h)
{{article_taglist}}	Filled with a comma separated list of tags.

marker	description
Specific for the element recentrecord.h **Common markers apply, in addition the following specific markers are recognized:**	
{{article_link}}	Filled with the link to the article (URL encoded).
{{article_title}}	Filled with the article's title. CAUTION: this value is converted with UTF8 named character entities for characters above ASCII, therefore it cannot be used inside attributes.
{{article_abstract}}	Filled with the abstract of the article, the abstract is converted from text marker mode to HTML. Links, images and other elements beside text should not be used in abstract.
{{article_poster}}	Replaced with the element postersnippet.h
{{article_posterURL}}	Filled with the image poster URL. Remark: This is redundant if {{article_poster}} is used.
{{article_poster_depiction}}	Filled with the depiction (*alt* attribute) of the image. Remark: This is redundant if {{article_poster}} is used.
{{article_video}}	Replaced with the element videolink.h or videosnippet.h in the case the Video snippet checkbox is checked.
{{article_hasvideoicon}}	Filled with the URL to the image video icon as specified in Icon indicator for articles with video in **Site Settings**. If the article does not have a video associated or the icon has not been defined in Site Settings, this marker is removed.
{{article_category}}	Filled with category name associated with the article. CAUTION: this value is converted with UTF8 named character entities for characters above ASCII, therefore it cannot be used inside attributes.
{{URL_article_category}}	Filled with URL encoded category name associated with the article.
{{article_date}} {{article_time}}	Respectively, last modified date and time of the Article.
{{created_date}} {{created_time}}	Respectively, creation date and time of the Article.

marker	description
{{article_author}} {{article_coauthor}} {{article_s_author}} {{article_s_coauthor}}	Author name, as specified in **Site Settings** if no Author is specified in the article's attributes on the left pane.
{{article_tags}}	Filled with the tag records (see tagrecord.h)
{{article_taglist}}	Filled with a comma separated list of tags.

marker	description
Specific for the element categoryrecord.h **Common markers apply, in addition the following specific markers are recognized:**	
{{categoryname}}	Filled with the name of the category. CAUTION: this value is converted with UTF8 named character entities for characters above ASCII, therefore it cannot be used inside attributes.
{{URLcategoryname}}	Filled with the URL encoded name of the category.

marker	description
Specific for the element tagrecord.h **Common markers apply, in addition the following specific markers are recognized:**	
{{tagname}}	Filled with the name of the tag. CAUTION: this value is converted with UTF8 named character entities for characters above ASCII, therefore it cannot be used inside attributes.
{{URLtagname}}	Filled with the URL encoded name of the tag.

Enumerated and Named Styles

Inside the file bm_styles.css some styles can be recognized by BlogManager to load them into the Format > Style and Format > Named style menu.

Use a special comment to instruct BlogManager:
`/*{{menu:`***`styletype`***`;desc:`***`description`***`;name:`***`stylename`***`}}*/`

There must be no spaces between /* and {{ and bwteen }} and */ . The last part (colored in blue) should be omitted for *styletype* = **style**.

Styletype must be **`style`**, **`namedstyle`** or **`namedstyleblock`**
This specify the menu and method to generate the style marks.

style Reserved for enumerated styles. Translated into `<span>` .
These are restricted to `bm_enumstyle_0` to `bm_enumstyle_9` .
Do not specify *name:stylename* for these type of styles.

namedstyle Reserved for named styles, these are translated into `<span>` .
These should be defined as `bm_n_`*`yourname`*`_` where *yourname* is a name you give (max 20 characters), this name must be used in place of ***`stylename`***.

namedstyleblock
Reserved for styled blocks of text, translated into `<div>` .
These can have any name (max 20 characters).
The name given must be used in place of ***`stylename`***.

Description Enter any maeningful description. It is recommended to not exceed 50 characters. This will appear on the menu.

Stylename Enter the name you gave to the style. For namedstyle it is the name in between `bm_n_` and the final underscore _ . For namestyleblock it is the whole name given to the style.

Example:

```
.bmtextbox {display:block;width:80%;margin:auto;background-color:#e8e8e8;padding:2px;}
/*{{menu:namedstyleblock;desc:text box;name:bmtextbox}}*/
```

Built-in Reserved Styles

Following are described some built-in, preset, reserved class styles automatically applied to some elements.

class style	description
`bm_table_` `bm_table_>div` `bm_table_>div>div`	Specify the appearance of a table of elements. .bm_table_ > div>div should be defined as well to specify the appearance of the cells. .bm_table_>div should be defined to specify the rows.
`bm_bordertable_` `bm_bordertable_>div` `bm_bordertable_>div>div`	As above but for tables with borders.
`bm_thbold_`	Specify the a bold/heavy table heading marked with double line (equal signs), example: `Column bold header` `==========` Applied to internally generated elements.
`bm_ththin_`	Specify the table thin/light table heading marked with a line of dashes, example: `Column thin header` `----------` Remarks: Applied to internally generated elements. Invisible table heading do not requires any style as it is classed as normal sub or a bm_table_ class.
`bm_center_`	Specify the paragraph should be centered, applied to internally generated <p> elements that are created when `[:c:]` is at the beginning of a line.
`bm_a_`	Specify the appearance of the <a> entities created by the automatic detection of the strings that begins with http.
`bm_tab`	Specify the margin left of a element that is created in place of tabulation characters (TAB) if they are found from the beginning of the line.

class style	description
bm_enumstyle_x *where x is 0 to 9:* bm_enumstyle_0 *to* bm_enumstyle_9 *may follow description:* /*{{desc: invert color}}*/	Specifies special enumerated styles, from 0 to 9. Applied to internally generated elements when a block of text is included within [x) and :] , where the *x* is a number between 0 to 9. Example: this [2)text will be in red:] if bm_enumstyle_2 is defined as: .bm_enumstyle_0 { color:red; } it will render as: this text will be in red To the style may follow a description enclosed in a css comment, but recognized by the program. It must begin by /*{{desc: and a description of the style may ensue. The description must be ended by }}*/ This description will be shown in the Styles menu.
bm_n_<*name*>_ where name is any value entered in text.	Specifies a named style (i.e., color) entered in the text marker. The name should be shorter than 20 characters. Example: some text [:_enh_:enhanced text:] normal again Requires bm_n_enh be defined: bm_n_ehn_ {background-color:yellow} So that it will render as: some text enhanced text normal again Applies to internally generated elements.
bm_code_mark bm_code_lang bm_code_lanB	Style for markup languages, for C/C++/Java/javascript style languages and for Basic style languages, respectively.
bm_code_error bm_code_highlight bm_code_break bm_code_ref	Stylize content inside a code element. They are used with [! !] , [* *] , [? ?] and [^ ^] respectively.
bm_figures_	Specify the style for figures. It applies to <figure> elements, that contain a <img> element and a <figcaption> element, therefore the following styles can be specified as well: bm_figures_ > img bm_figures_ > figcaption
bm_articlecategory_	Specify the style for a record of a category.

class style	description
bm_articletag_	Specify the style for a record of a tag.
bm_searchtitle_	Specify the style of an element holding the title and link to the article as a result in the search page. Applies to .
bm_searchinfo_	Specify the style of an element holding tags, category and date that contains: divs that have classes bm_searchct_ and bm_searchmodified_ (see below). Applies to
bm_searchct_	Specify the style of elements holding the category or tags.
bm_searchmodified_	Specify the style of an element holding last modified date
bm_noteref	Style for the footnote reference. The footnote reference is the number enclosed in brackets followed by colon, example: `[3]: note about text.`
bm_notelnk	Style for the note link. This is the bracketed number that may follow a text line to cite a reference in footnotes, example: `some text[3] more text`
bm_videos_	Style for the elements <video> used in the main body content of the page/article (not to be confused with main video or video banner).
bm_mainvideo_	Style the main video put in place of the article's poster.
bm_vbanner_	Style for the element <video> used as video banner. This video replaces the banner of the page.

In addition to the aforementioned classes the following elements can be stylized:

<p>	Paragraphs generated within a <section> .
<section>	Sections created by a series of consecutive equal signs from the beginning of a line.
and	Lists generated

Remarks

Further elements are defined in the templates and it is up to you to stylize them as you like.