

Blorg Manual

Release 0.71

by Bastien Guerry

This manual is for Blorg (version 0.71).

Copyright © 2006 Bastien Guerry

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with no Invariant Sections, with the Front-Cover texts being “A GNU Manual,” and with the Back-Cover Texts as in (a) below. A copy of the license is included in the section entitled “GNU Free Documentation License.”

(a) The FSF’s Back-Cover Text is: “You have freedom to copy and modify this GNU Manual, like GNU software. Copies published by the Free Software Foundation raise funds for GNU development.”

Table of Contents

1	Quickstart	1
1.1	Another blog engine?	1
1.2	Install blorg	1
1.3	Example	1
1.4	Publishing process	2
	blorg-publish	2
	Rendering steps	2
1.5	Configuration	3
2	Blorg File	4
2.1	The header	4
2.2	Text Formatting Rules	5
3	Templates	6
3.1	Blog Structure	6
3.2	HTML Structure	6
3.3	Index page	7
3.4	Post Template	8
	3.4.1 Templates in templates	9
	3.4.2 Full post display	9
	3.4.3 Similar options	9
4	Options	10
	Index	11

1 Quickstart

Blorg is a tool that helps you to produce a blog based on an org-mode file. If you're here, there are good chances that you've heard about Org-mode. If you don't, please check Org-mode before : <http://staff.science.uva.nl/~dominik/Tools/org/>

Note that blorg has been developed under GNU Emacs. It should work fine under XEmacs, but it has not been thoroughly tested yet.

1.1 Another blog engine?

Blorg is another blog publishing tool. There are already many of them in the wild, so why would you prefer this one?

- Spend time **blogging**, not installing blogs.
- Blorg is an **Emacs** library and Emacs is your favorite editor.
- Blorg is based on **org-mode** and org-mode is your favorite organizer.
- **Control** the HTML structure of your pages. Keep it simple. Keep it CSS-friendly.
- You like **tags**. You want to be able to create, modify and update them **on the fly**.
- Publishing a blog post should be as **easy** as hitting a key. Well, now it is.
- You don't want to take the burden of installing some Php/MySQL-based blog.
- You don't need features like comments and trackbacks.

1.2 Install blorg

Get 'blorg.el': <http://www.cognition.ens.fr/~guerry/u/blorg.el>

Then make sure 'blorg.el' is in your `load-path` and require it somewhere in your configuration.

For example, if 'blorg.el' is in the `~/elisp/` directory, the following lines should do the trick:

```
(add-to-list 'load-path "~/elisp/")
(require 'blorg)
```

1.3 Example

Blorg files are just like any other org file except that they understand a few specific keywords.

Here is a minimalistic example of what an blorg file looks like. Copy this example somewhere and try `C-c "` (blorg-publish).

```
#+TITLE:      The title of my blog
#+BLOG_URL:   http://www.server.org/~me/
#+PUBLISH_DIR: /home/me/public_html/

* DONE This is my first entry           :misc:
  CLOSED: [2006-11-03 ven 16:57]

Hello world!
```

Please make sure to adapt this example to your environment before trying it! Let's explain the structure of this example a bit.

`#+TITLE:`

`#+BLOG_URL:`

`#+PUBLISH_DIR:`

These keywords respectively define the title of the blog, the full URL of the blog (as publicly available from the web) and the server directory where to upload files.

See [Section 2.1 \[The header\], page 4](#) for further details about available keywords in `blorg`.

`* DONE This is my first entry :misc:`

The title of the first blog post. Only first-level headlines marked as `DONE` are considered as suitable for publication. Headlines containing other `TODO` keywords are simply ignored.

For a full description of all available variables in the header, see [Section 2.1 \[The header\], page 4](#).

`:misc:` is what `org-mode` defines as a *tag*. it will be used by `blorg` to publish two files: `'misc.html'` will contain all posts that are tagged as `misc` and `'misc.xml'` will contain a feed for this tag.

`CLOSED: [2006-11-03 ven 16:57]`

Creation date of the post. Creation date is the date when you first *locally* finished or updated the post. Note that this is not always the same as the *publication* date, since publication date is the date when you actually upload the files on the server.

For this `CLOSED:` string to be automatically called when finishing a blog entry, turn on `'org-log-done'`.

1.4 Publishing process

`blorg-publish`

Calling `blorg-publish (C-c ")` parse the `blorg` file and export it as a blog.

By default, `blorg` does not re-publish a post page if this page already exists on the server. You can force the upload of all post pages by prefixing `blorg-publish: C-u C-c "`.

Giving a numeric argument will force the index to display the given number of posts and force the upload of all post page. `C-u 4 C-c "` makes `blorg` list 4 posts in the index page.

Rendering steps

`Blorg` first parses the header and the content of the `blorg` file. Then the file is rendered as a blog. The content of the blog is processed in this order.

1. Check whether the `'upload'` and `'images'` directories do exist and create them if not.
2. Render the **index page**. Create an atom or rss feed for this page if `index` is an element of `'blorg-publish-feed'`.
3. Render the **tag page(s)**. Create an atom or rss feed for each tag page if `tag` is an element of `'blorg-publish-feed'`.

4. Render **month page(s)**. No feed is published for month pages, for it doesn't make sense. Maybe force re-publishing of all month pages, depending on whether `blorg-publish` is prefixed or not.
5. Render **post page(s)**.

1.5 Configuration

There are mainly three ways of configuring blorg.

1. Use the traditionnal Customize interface (`M-x customize-group RET blorg`).
2. Use separate emacs-lisp file for the whole system: see '`blorg-config-file`'.
3. Use a separate emacs-lisp config file for a specific blorg file by giving a value to the `#+CONFIG_FILE` keyword.

2 Blog File

This section describes the file structure. More precisely it goes through available keywords that you can put in the header and it describes how blog handles text formatting.

2.1 The header

In org-mode, the header is defined by the first lines of the file, each line containing a keyword. This section describes all available keywords in blog. Each keyword locally assigns a value to the corresponding variable. If you want to modify those variables globally, customize ‘blog-default-options’.

Keywords marked with * are not known by org-mode (but the * is *not* part of the keyword.)

#+TITLE, #+BLOG_URL*, #+PUBLISH_DIR*

These keywords are **mandatory**. They respectively define the title of the blog, the full URL of the blog (as publicly available from the web) and the server directory where to upload files.

#+AUTHOR, #+EMAIL, #+LANGUAGE

These keywords have the same meaning in org-mode and in blog. They are not mandatory, but they represent the core information you generally have to provide for a blog.

#+AUTHOR defaults to `user-mail-address`.

#+EMAIL defaults to `user-full-name`.

#+LANGAGE defaults to the first two characters of the LANG environment variable.

#+ENCODING*

Defines the encoding of the buffer and the resulting encoding of HTML pages.

#+UPLOAD_DIR*, #+IMAGES_DIR*

Defines the relative upload and images directories. The default for these is `upload/` and `images/`.

#+FEED_TYPE*, #+HTML_CSS*, #+XML_CSS*

#+FEED_TYPE can be either `atom` or `rss`, depending on whether you prefer Atom 1.0 or RSS 2.0 as the format for your XML feeds.

#+HTML_CSS defines the CSS stylesheet for the HTML files.

#+XML_CSS defines the CSS stylesheet for the XML feed. This is now quite obsolete since many browsers (e.g. Firefox) now have a special handling of XML feeds.

#+SUBTITLE*, #+HOMEPAGE*, #+KEYWORDS*

The subtitle might be useful in some occasions. See [Chapter 3 \[Templates\], page 6](#) to see how and when. The homepage is the homepage of the author, if available.

#+KEYWORDS are the keywords that will take inside the `<meta name="keywords" />` of the HTML output.

#+CREATED*

Define the date creation of the blog.

#+DONE_STRING*

Usually, the last element of `org-todo-keywords` is `DONE`. `blog` will use it as the default keyword for headlines that are suitable for publication. If `#+SEQ_TODO` is defined, this will override `org-todo-keywords`.

If `#+DONE_STRING` is defined, it will be used as a special string marking publishable headlines.

Be aware that if you use a `#+DONE_STRING` that is different from the traditional `DONE` keyword, you will get trouble logging headlines with `CLOSED:` string.

Defines the number of posts to process when exporting the index and the tags pages.

Other keywords like `#+STARTUP`, `#+LINK`, `#+TBLFM`, `#+SEQ_TODO`, `#+CATEGORY`, `#+TAGS` have no special meaning in `blog`.

2.2 Text Formatting Rules

Usual text formatting will be exported without complain.

- Sectioning
- Environments
- Fontification
- Images

3 Templates

This section describes the templates mechanism as it is implemented in blorg. Templates allow you to make the HTML output of blorg look the way you want.

- blorg-tag-page-template
- blorg-month-page-template
- blorg-post-page-template
- blorg-post-template
- blorg-post-author-template
- blorg-post-dates-template
- blorg-post-tags-template
- blorg-post-tags-template

3.1 Blog Structure

Here's an overall view of the blog structure :

- Feeds
- Index page
- Posts pages
- Months pages
- Tags pages
- Upload and images dir

3.2 HTML Structure

Here is the default hierarchy for the HTML page. ‘Included tag’ describes the first tag to appear in the element (if any). ‘In template’ says whether the corresponding HTML element is by default included in templates.

‘*’ means it’s included in *all* templates. ‘!page’ means it’s *not* included in the ‘page’ template. ‘!*’ means it’s not included in any template.

HTML element	included tag	in templates
HTML	-	*
.BODY	-	*
..HEAD	-	*
...DIV id:content	-	*
....DIV id:blog-title	<h1 />	*
....DIV id:blog-author	<h3 />	*
....DIV id:tags-cloud	<a />	!post
....DIV id:prev-posts		!post
....DIV id:archives		!post

```

....DIV class:post           -           *
.....DIV class:post-title   -           *
.....DIV class:post-infos   -           *
.....P class:author         -           *
.....P class:date           -           *
.....P class:tags          -           *
.....DIV class:post-echos    -           !*
.....DIV class:post-content -           *
.....A class:read-more      -           !post

```

In the next two sections, we get a closer look at two templates: the `index` template and the `post` template, respectively defined by `'blorg-index-template'` and `'blorg-post-template'`.

3.3 Index page

Each template is a mix of HTML code and emacs-lisp expressions.

Here is `blorg-index-template`:

```

<body>
<div id="content">

<div id="blog-title">
  <h1><a href="(blorg-insert-index-url)"> \
    (blorg-insert-page-title)</a></h1>
</div>

<div id="blog-author">
  <h3><a href="(blorg-insert-mailto-email)"> \
    (blorg-insert-author)</a></h3>
</div>

(blorg-insert-previous-posts)
(blorg-insert-tags-as-cloud)
(blorg-insert-archives)
(blorg-insert-content)

</div>
</body>

```

Depending on the template, you have a set of available elisp expressions that you can insert in the template. Check the docstring of each template variable to get the list of available expression.

Here is the list of available elisp defuns you can insert in the `index` template:

```

(blorg-insert-index-url)      : the URL of the index page
(blorg-insert-homepage)      : the URL of the author's homepage
(blorg-insert-page-title)    : the page title
(blorg-insert-page-subtitle) : the page subtitle
(blorg-insert-mailto-email)  : mailto:your@email.com

```

```

(blog-insert-email)           : your@email.com
(blog-insert-author)         : author's name
(blog-insert-previous-posts) : a list of previous posts
(blog-insert-tags-as-cloud)  : a cloud of tags
(blog-insert-tags-as-list)   : a list of tags
(blog-insert-archives)       : a list of archived months
(blog-insert-content)        : the main content

```

3.4 Post Template

The post template defines the display of a post in a page.

Attention: we're not talking about the way each post-page is structured (which is defined by 'blog-post-page-template') but the way each post appears in a [index/month/tag/post]-page.

Here is the default template for posts:

```

<div class="post">

  <div class="post-title">
    <h2><a href="(blog-insert-post-url)"> \
      (blog-insert-post-title)</a></h2>
  </div>

  <div class="post-infos">
    (blog-insert-post-author)
    (blog-insert-post-dates)
    (blog-insert-post-tags)
  </div>

  <div class="post-content">
    (blog-insert-post-content)
  </div>

</div>

```

Here is the list of available elisp defuns you can insert in the post template:

```

(blog-insert-post-url)       : the URL of the post
(blog-insert-post-title)     : the title of the post
(blog-insert-post-author)    : the author of the post
(blog-insert-post-dates)     : the publication dates
(blog-insert-post-tags)      : the tags for this post
(blog-insert-post-echos)     : the "Submit this post" links
(blog-insert-post-content)   : the main content of the post

```

Please note that this list slightly differs from that of the index template. The list of potential elements is not exactly the same for each template.

3.4.1 Templates in templates

The funny thing is that even in the post template, there are more (hidden) templates. For example, the display of `blorg-insert-post-author` is controlled by `'blorg-post-author-template'`. See `'blorg-post-dates-template'` and `'blorg-post-tags-template'`.

3.4.2 Full post display

Let's say that you want the full post to be displayed in every page but in the index. You can do this by setting `'blorg-put-full-post'`. This option is a list of symbols among `index`, `post`, `tag` and `month`. The default is `'(index)'` so that only post pages display the full content of posts.

3.4.3 Similar options

Here is a list of other options working the same way:

- `blorg-put-author-in-post`
- `blorg-put-dates-in-post`
- `blorg-put-tags-in-post`
- `blorg-put-echos-in-post`

All these options override the display of a post as set in `'blorg-post-template'`. And in fact, you might say they are redundant since the structure of the post is already defined in this template. But

1. it's **easier** and **faster** to edit such options than to edit the whole template;
2. separate options let you distinguish between the templates (that *globally* apply to the system) and the file-specific options. You can add file-specific options in a separate config file. The name of this file is set by `'org-bloggging-config-file'` and/or in the header with the keyword `#+CONFIG_FILE`.

4 Options

Index

#

#+AUTHOR.....	4
#+BLOG_URL.....	2, 4
#+CONFIG_FILE.....	3, 9
#+CREATED.....	5
#+DONE_STRING.....	5
#+EMAIL.....	4
#+ENCODING.....	4
#+FEED_TYPE.....	4
#+HOMEPAGE.....	4
#+HTML_CSS.....	4
#+IMAGES_DIR.....	4
#+KEYWORDS.....	4
#+LANGUAGE.....	4
#+PUBLISH_DIR.....	2, 4
#+SUBTITLE.....	4
#+TITLE.....	2, 4
#+UPLOAD_DIR.....	4
#+XML_CSS.....	4

A

atom.....	2
-----------	---

B

blog.....	1
blorg-config-file.....	3
blorg-default-options.....	4
blorg-post-author-template.....	9
blorg-post-dates-template.....	9
blorg-post-number-per-page.....	2
blorg-post-tags-template.....	9
blorg-post-template.....	9
blorg-publish.....	1
blorg-put-author-in-post.....	9
blorg-put-dates-in-post.....	9
blorg-put-echos-in-post.....	9
blorg-put-tags-in-post.....	9

C

C-c ".....	1, 2
C-u C-c ".....	1, 2
CLOSED.....	2
customize.....	3

D

date.....	2
DONE.....	2

F

feed.....	2
full post.....	9

I

index.....	9
------------	---

M

month.....	9
------------	---

O

org-log-done.....	2
-------------------	---

P

post.....	9
-----------	---

R

rss.....	2
----------	---

T

tag.....	2, 9
----------	------