Discount

Discount is free software released under the terms of a BSD-style license.

If you find it useful, please consider making a contribution to help support onward development.

Donate

DOWNLOAD

Discount 2.1.6, released 16-Mar-2013.

DESCRIPTION

This is my implementation of John Gruber's Markdown text to html language. There's not much here that differentiates it from any of the existing Markdown implementations except that it's written in C instead of one of the vast flock of scripting languages that are fighting it out for the Perl crown.

Markdown provides a library that gives you formatting functions suitable for marking down entire documents or lines of text, a command-line program that you can use to mark down documents interactively or from a script, and a tiny (3 programs so far) suite of example programs that show how to fully utilize the markdown library.

My markdown also does, by default, various smartypants-style substitutions.

The program

The `markdown` program is a trivial compiler that reads in a markdown file and writes out a html document or – if you use the `-d` flag – an outline showing the parse tree. It does have a few options;

- `-d` is, as previously mentioned, the flag that makes markdown produce a parse tree instead of a html document.
- `-F <flags>` sets various flags that change how markdown works. The flags argument is a somewhat less than obvious bitmask – for example, `-F 0x4` tells `markdown` to **not** do the smartypants translations on the output. (there are cases – like running the test suite – where this is a useful feature.)
- `-o file` tells markdown to write the output to *`file`*
- `-V` tells you a markdown version number and how the package was configured. For example

  ```
  $ markdown -V
  markdown: discount 1.0.0 DL_TAG HEADER TAB=8
  ```

  tells you that this is markdown 1.0.0, and that the package was configured with support for definition lists, pandoc-style document headers, and sensible tabs.

The library

There are 17 public functions in the markdown library, broken into three categories:

Input functions

1. `MMIOT *mkd_in(FILE *f, int flags)` reads a markdown input file and returns a MMIOT containing the preprocessed document. (which is then fed to `markdown()` for final formatting.)

2. `MMIOT *mkd_string(char *bfr, int size, int flags)` reads the markdown input file that's been written into `bfr` and returns a preprocessed blob suitable for feedin to `markdown()`. This function exists because annotations uses `mmap()` to access message files instead of traditional file i/o. (If you're going to port Markdown to an AS/400, this function *is* the droid you've been looking for.)

"Big Picture"-style processing functions

1. `int markdown(MMIOT *doc, FILE *out, int flags)` formats a document (created with `mkd_in()` or `mkd_string()`) and writes the resulting HTML document to `out`.

2. `int mkd_line(char *bfr, int size, char **out, int flags)` allocates a buffer, then formats the text string into that buffer. text string, allocates a buffer, The differences from `markdown()` are it doesn't support quoting, footnotes ("reference links",) multiple paragraphs, lists, code sections, or pure html sections.

3. `int mkd_generateline(char*bfr, int size, FILE *out, int flags)` formats the text string and writes the resulting HTML fragment to `out`. It is exactly like `mkd_line()` except that it writes the output to a `FILE*`.

Fine-grained access to the internals

1. `int mkd_compile(MMIOT *doc, int flags)` takes a document created by `mkd_in()` or `mkd_string()` and compiles it into a tree of block elements.

2. `int mkd_generatehtml(MMIOT *doc, FILE *out)` generates html from a compiled document.

3. `int mkd_document(MMIOT *doc, char **text)` returns (in `text`) a pointer to the compiled html document, and (in the return code) the size of that document.

4. `int mkd_css(MMIOT *doc, char **out)` allocates a buffer and populates it with any style blocks found in the document.

5. `int mkd_generatecss(MMIOT *doc, FILE *out)` prints any style blocks in the document.

6. `int mkd_toc(MMIOT *doc, char **out)` allocates a buffer, populates it with a table of contents, assigns it to out, and returns the length of the buffer.

   To get a table of contents, you must `compile()` the document with the `MKD_TOC` flag (described below)

7. `int mkd_generatetoc(MMIOT *doc, FILE *out)` writes a table of contents to *out*; other than writing to a `FILE*`, it operates exactly like `mkd_toc()`

8. `int mkd_dump(MMIOT *doc, FILE *f, int flags, char *title)` prints a block structure diagram of a compiled document.

9. `void mkd_cleanup(MMIOT *doc)` releases the MMIOT allocated for the document.

Document header access functions

1. `char *mkd_doc_title(MMIOT *doc)` returns the `% title` line.
2. `char *mkd_doc_author(MMIOT *doc)` returns the `% author(s)` line.
3. `char *mkd_doc_date(MMIOT *doc)` returns the `% date` line.

Url callback functions

1. `void mkd_e_url(MMIOT*, char* (callback)(char*,int,void*))` sets up a callback function that is called whenever discount processes a `[]()` or `<link>` construct. The callback function is passed a pointer to the url, the size of the url, and a data pointer (null or supplied by `mkd_e_data()`)
2. `void mkd_e_flags(MMIOT*, char *(callback)(char*,int,void*))` sets up a callback to provide additional arguments to the tags generated by `[]()` and `<link>` constructs. If, for instance, you wanted to add `target="_blank"` to every generated url, you could just make a callback function that returned that string.
3. `void mkd_e_free(char *, void*)` is called to free any allocated memory returned by the url or flags callbacks.
4. `void mkd_e_data(MMIOT*, void*)` assigns a callback data area to the url & flags callbacks.

The flags argument in `markdown()`, `mkd_text()`, `mkd_in()`, `mkd_string()`, `mkd_compile()`, and `mkd_generatehtml()` is a mask of the following flag bits:

Flag	Action
MKD_NOLINKS	Don't do link processing, block `<a>` tags
MKD_NOIMAGE	Don't do image processing, block `<img>`

MKD_NOPANTS	Don’t run `smartypants()`
MKD_NOHTML	Don’t allow raw html through **AT ALL**
MKD_STRICT	Disable `SUPERSCRIPT`, `RELAXED_EMPHASIS`
MKD_TAGTEXT	Process text inside an html tag; no `<em>`, no `<bold>`, no html or `[]` expansion
MKD_NO_EXT	Don’t allow pseudo-protocols
MKD_CDATA	Generate code for xml `![CDATA[...]]`
MKD_NOSUPERSCRIPT	No `A^B`
MKD_NORELAXED	Emphasis happens *everywhere*
MKD_NOTABLES	Don’t process PHP Markdown Extra tables.
MKD_NOSTRIKETHROUGH	Forbid `~~strikethrough~~`
MKD_TOC	Do table-of-contents processing
MKD_1_COMPAT	Compatability with MarkdownTest_1.0
MKD_AUTOLINK	Make `http://foo.com` a link even without `<>`s
MKD_SAFELINK	Paranoid check for link protocol
MKD_NOHEADER	Don’t process document headers
MKD_TABSTOP	Expand tabs to 4 spaces
MKD_NODIVQUOTE	Forbid `>%class%` blocks
MKD_NOALPHALIST	Forbid alphabetic lists
MKD_NODLIST	Forbid definition lists
MKD_EXTRA_FOOTNOTE	Enable PHP Markdown Extra-style footnotes.

Smartypants substitutions

1. `` `` text `` \‘\’ is translated to “text”.
2. `"double-quoted text"` becomes “double-quoted text”
3. `'single-quoted text'` becomes ‘single-quoted text’
4. `don't` is “don’t.” as well as anything-else’t. (But `foo'tbar` is just foo'tbar.)
5. And `it's` is “it’s,” as well as anything-else’s (except not foo'sbar and the like.)
6. `(tm)` becomes ™
7. `(r)` becomes ®
8. `(c)` becomes ©
9. `1/4th` ? ¼th. Ditto for `1/4` (¼), `1/2` (½), `3/4ths` (¾ths), and `3/4` (¾).
10. ... becomes …
11. . . . also becomes …
12. --- becomes —
13. -- becomes –

14. `A^B` becomes A^B. Complex superscripts can be enclosed in `()`s, so `A^(B+2)` becomes A^{B+2}.

Language extensions

My markdown was written so I could replace the fairly gross homemade text to html prettifier that I wrote for annotations, so I've extended it in a few ways; I've put support for paragraph centering in so that I don't have to hand enter the `<center>` and `</center>` tags (nowadays I generate a css-styled `<p>` block, because that's xhtml compatible instead of the now-depreciated `<center>` block element.) I've added support for specifying image sizes, and I've written a not-earthshatteringly-horrible markup extension for definition lists.

Paragraph centering:

To center a paragraph, frame it with `->` and `<-`.

```
->this is a test<-
```

produces

this is a test

Specifying image sizes:

An image size is defined by adding an additional =*width*x*height* field to the image tag:

```
![dust mite](http://dust.mite =150x150)
```

produces

Definition lists:

To mark up a definition list, left-justify the label and frame it with `=` characters, then put the body of the list item on the next line, indented 4 spaces.

```
=hey!=
    This is a definition list
```

produces

hey!: This is a definition list

A definition list label is just a regular line of markdown code, so you can put links and images into it.

In discount 1.2.3, the definition list syntax has been extended so that you can define sequential `<dt>` blocks by doing

```
=tag1=
=tag2=
    data.
```

which generates

```
<dt>tag1</dt>
<dt>tag2</dt>
<dd>data.</dd>
```

In discount 2.0.4 I extended the definition list syntax to allow php markdown extra definition lists which means that source like

```
tag1
: data
```

now generates

```
<dt>tag1</dt>
<dd>data</dd>
```

alpha lists: Ordered lists with alphabetic labels (enabled by `--enable-alpha-list` during configuration) are supported in the same way that numeric ordered lists are:

```
a. first item
b. second item
```

generates

a. first item
b. second item

New pseudo-protocols for [] links: I wanted to be able to apply styles inline without having to manually enter the `<span class="`*xxx*`">...</span>` html. So I redid the `[][]` code to support some new “protocols” within my markdown:

`abbr:`*description*:	The label will be wrapped by `<abbr title="`*description*`">...</abbr>`
`class:`*name*:	The label will be wrapped by `<span class="`*name*`">...</span>`
`id:`*name*:	The label will be wrapped by `<a id="`*name*`">...</a>`
`raw:`*text*:	*Text* will be written verbatim to the output. The protocol was inspired by a short thread on the markdown mailing list about someone wanting to embed LaTeX inside `<!-- -->` and finding, to their distress, that markdown mangled it. Passing text through in comments seems to be a path to unreadable madness, so I didn't want to do that. This is, to my mind, a better solution.

Style blocks: accept `<style>...</style>` blocks and set them aside for printing via `mkd_style()`.

Class blocks: A blockquote with a first line of `> %class%` will become `<div class="`*class*`">` instead of a `<blockquote>`.

Tables: PHP Markdown Extra-style tables are supported;

```
 aaa | bbbb
-----|------
hello|sailor
```

becomes the following table:

aaa	**bbbb**
hello	sailor

And much of the rest of the current table syntax (alignment, handling of orphan columns) follows the PHP Markdown Extra spec.

Document Headers:

[Pandoc]-style document headers are supported; if the first three lines in the document begin with a `%` character, they are taken to be a document header in the form of

```
% Document title
% Document author
% Document date
```

and can be retrieved by the library functions `mkd_doc_title()`, `mkd_doc_author()`, and `mkd_doc_date()`.

Note that I implement Pandoc document headers as they were documented in 2008; any Pandoc changes since then will not be reflected in my implementation.

Fenced code blocks:

If configured with the `--with-fenced-code` options, Pandoc-style fenced code blocks are supported; blocks of code wrapped in ~~~ lines are treated as code just as if it was indented the traditional 4 spaces. Github-flavored-markdown fenced code blocks (blocks wrapped in backtick lines) are also supported.

Neither of these formats support the github-flavored-markdown class extension where you can put a word at the end of the opening backtick line and have the block given that class.

How standard is it?

When I run the standard test suite (version 1.0.3) from daringfireball, `MarkdownTest.pl` reports:

```
$ MARKDOWN_FLAGS=0x20004 ./MarkdownTest.pl --tidy --script=/usr/local/bin/markdown
Amps and angle encoding ... OK
Auto links ... OK
Backslash escapes ... OK
Blockquotes with code blocks ... OK
Code Blocks ... OK
Code Spans ... OK
Hard-wrapped paragraphs with list-like lines ... OK
Horizontal rules ... OK
Inline HTML (Advanced) ... OK
Inline HTML (Simple) ... OK
Inline HTML comments ... OK
Links, inline style ... OK
Links, reference style ... OK
Links, shortcut references ... OK
Literal quotes in titles ... OK
```

```
Markdown Documentation - Basics ... OK
Markdown Documentation - Syntax ... OK
Nested blockquotes ... OK
Ordered and unordered lists ... OK
Strong and em together ... OK
Tabs ... OK
Tidyness ... OK


22 passed; 0 failed.
```

When I run the old standard test suite from daringfireball, `MarkdownTest.pl` reports:

```
$ MARKDOWN_FLAGS=0x22004 ./MarkdownTest.pl --tidy --script=/usr/local/bin/markdown
Amps and angle encoding ... OK
Auto links ... OK
Backslash escapes ... OK
Blockquotes with code blocks ... OK
Hard-wrapped paragraphs with list-like lines ... OK
Horizontal rules ... OK
Inline HTML (Advanced) ... OK
Inline HTML (Simple) ... OK
Inline HTML comments ... OK
Links, inline style ... OK
Links, reference style ... OK
Literal quotes in titles ... OK
Markdown Documentation - Basics ... OK
Markdown Documentation - Syntax ... OK
Nested blockquotes ... OK
Ordered and unordered lists ... OK
Strong and em together ... OK
Tabs ... OK
Tidyness ... OK


19 passed; 0 failed.
```

Most of the “how to get standards compliant” changes that went in were cleaning up corner cases and blatant misreading of the spec, but there were two places where I had to do a horrible hack to get compliant:

1. To pass the **Hard-wrapped paragraphs with list-like lines** test, I had to modify `mkd_compile()` so that it would have top-level paragraphs absorb adjacent list items, but I had to retain the old (and, IMO, *correct*) behavior of a new list forcing a block break within indented (quoted, inside lists) blocks..
2. To pass the **Markdown Documentation - Syntax** test in MarkdownTest 1.0, I had to change the behavior of code block from “preserve trailing whitespace” to “preserve trailing whitespace *unless* it’s the first line in the block.” From version 1.3.3 on, this is no longer the default, but the flag `MKD_1_COMPAT` (0x2000) turns it on again for testing purposes.

Does this markdown treat tabs as 4 spaces?

By default, yes, it does. The habit of compensating for broken editors that give no way to indent except for tabbing by setting tabstops to 4 is so intertwined with this language that treating tabs properly would be the moral equivalent of dropping nuclear devices into the testsuite.

But if you use a proper tabstop (8 characters), you can configure markdown with **`--with-tabstop`** and it will expand tabs to 8 spaces. If you've configured your markdown like this (`markdown -V` will report **`TAB=`**8) and you need to mark up text from other sources, you can set the input flag **`MKD_TABSTOP`** to revert those documents back to the icky standard 4-space tab.

Source Code

To build discount, untar your selected tarball, cd into the directory it creates, then do `configure.sh` to generate your Makefiles. After doing this, a `make` should give you a functional stack of programs and libraries.

Discount builds, for me, on SLS Linux, MacOS 10.5, FreeBSD 4.8, and RHEL3. It may build on Windows with mingw, but I'm not sure about that.

- version 2.1.6 does nothing except for some bugfixes (and ignores some particularly scary ones that I /must/ fix soon) and adds two small features.

  The bugfixes are:

  1. A < at the end of the input is exactly the same as <(space)
  2. Markdown.pl does not appear to escape `\<[nonwhite]` sequences. Sigh.
  3. Tweak the previous `Markdown does not escape...` commit to simply push out the backslash and back up to the start of the `<[nonwhite]` sequence, so -fnohtml will continue to work.
  4. Treat hard `<br/>` (via two spaces) as whitespace.
  5. Tweak divquote handling so that two adjacent divquotes won't die if there is a space between the second > & leading %
  6. Tweak one of the list tests back to the previous behavior (I've put in a hack for list indentation, and accidentally committed the changes. Oops!)

  The features are that I now use styles for table cell alignment instead of `align=`, and that I'm using the 3-clause BSD license for this release (because there is one widely used closed-source license that claims that you can't dynamically link with code that uses the 4-clause license. Fine. I'll 3-clause this release to make the stupid GPL happy.)

- version 2.1.5a does *even* more cleanup to deal with clang, plus adds a few small features:
    - MKD_NOSTYLE – treat `<style>` blocks as regular html.
    - some github flavored markdown support; gfm_…() input methods that put hardbreaks (== two spaces) at the end of every input line.
    - support for github flavored markdown backtick-delimited code blocks (in addition to tilde-delimited codeblocks)
    - in the `markdown` program, add
        1. -S flag (tell markdown to spit out style sections)
        2. -n flag (tell markdown not to output generated text)

- version 2.1.3 cleans up a couple of bugs that have been here for a while, plus tweaks the build process a bit for better compilation with the LLVM C compiler, mingw, and cygwin.

  The bugfixes are

    1. Stop tripping over tables with leading `|`s; the first implementation of tables treated leading `|`s as the end of an empty table cell. 2.1.3 corrects this to properly be merely decoration.

       As a side-effect, you now need to have all the rows of the table either with a leading `|` or not; you cannot mix and match them, sorry.

    2. For some mysterious reason I was treating the `<br>` tag as a block-level html tag, so my html blockifier would split paragraphs with explicit `<br>`s in them.

    3. The table of contents code was apparently generating bad html (something I never noticed, because I never use that feature, alas!) but *Stefano D'Angelo* contributed a patch to clean up the generated html to make it correct.

- version 2.1.2 tweaks table handling so that tables with leading `|`'s won't end up generating empty false `<td></td>`s, and that tables with trailing `|`'s won't end up getting those pipes included in the output.

- version 2.1.1.3 corrects a defect in smartypants processing that has been there since the beginning of time; I'd managed to misread the reference smartypants documentation and thought that **1** dash made an `&ndash;` and **2** dashes made an `&mdash;` (it's actually 2 and 3 dashes respectively.) *John Foerch* read the documentation recently, noticed it was wrong, and sent me a note pointing it out.

Whoops! But it's fixed now (as is the this page, which is, regrettably, the only documentation about smartypants.)

- version 2.1.1.2 corrects one small defect in block handling, plus changes the format of the output for failed tests in the test suite.

  The defect in block handling is that the reference implementation has text block absorbing adjacent code, so

  ```
  text text text
      code
  ```

  will generate

  ```
  <p>text text text code</p>
  ```

  instead of a paragraph of *text*(s) followed by `code`.

  The change in failed test output makes it output first the source code of the failed test, and then the *differences* between the expected output and the generated output.

- version 2.1.1.1 implements PHP markdown extra-style fenced code sections, where your chunks of code are surrounded by `~~~` lines instead of being indented 4 spaces.

  Fenced code sections are a *configuration* option, not a runtime option, and are enabled by using the `configure.sh` flag `--with-fenced-code` (**FENCED-CODE** in the version string).

  There are a few optimizations inside `markdown.c` to support fenced code detection without slowing the parser down, but those optimizations don't cause any of my test cases to fail. Version 2.1.1 is still a fairly experimental release despite this, so take some care with it.

- version 2.1.0 cleans up a small collection of warts and build bugs, plus adds some fairly small enhancements to the `theme` and `makepage` programs:

  - more modifications to `configure.sh` so that it generates *all* its test scripts in the build directory.
  - more MacOS support in `configure.sh`; check to see if `.dSYM` folders are created when a test script is compiled, and, if so, don't forget to delete them when they're cleaned up.
  - `makepage` now accepts markdown option flags a'la the `markdown` program (via `-Fxxxx`, `-fname`, or in the `MARKDOWN_FLAGS` environment variable.)

  - strip bitfields out of `opts[]` – I can't initialize a bitfield on plan9 cc.
  - add a `-E` flag to `theme` to ignore context-sensitivity on `<?theme xxx?>` substitutions.

- version 2.0.9 puts in a workaround for a Linux security misfeature; some of the commercial hosting services set up `/tmp` so that it's mounted `noexec`, which means that when `configure.sh` attempts to run a little test program to find scalar sizes, it can't do it and ends up failing. So `configure.sh` has been patched to put temporary executables that need to be executed into the working directory instead.

- version 2.0.8 introduces a small collection of patches, bugfixes, and performance enhancements:

  - Table processing was modified to make it hew closer to the markdown extra implementation
  - Optimized the code that's used to look for tables by scanning for pipe symbols when I'm reading data into the input lines array.
  - Documentation patches from David Banks
  - argument prototype tweaks for Solaris, from Allen Otis
  - `--with-github-tags` configuration option to to include - and _ in the acceptable characters for `maybe_tag_or_link()` (code from Ryan Tomayko's discount SCCS on github.)
  - in the `markdown` program, `-f?` gives a list of the known flags.
  - also, `-F?` gives a list of the known flag bits.
  - `--with-id-anchor` configuration options to have the table of contents code emit `id=` instead of the dummy `<a name=` anchor I replaced it with (`name=` is depreciated in xhtml, and may vanish.)

- version 2.0.7 finishes support for php markdown extra-style footnotes by documenting them, fixing one small initialization bug, and by creating the new public function `mkd_ref_prefix()` which changes the `name=` part of the forest of tags from `fn:N` and `fnref:N` to `{your prefix}:N` and `{your prefix}ref:N` (allows someone to footnote individual articles on a weblog where each article is rendered individually.)

  A few manpage typos are also corrected thanks to a patch from Alessandro Ghedini.

- version 2.0.6 adds preliminary (and undocumented, because I'm still peering at the published interface and internal data structures) support for php markdown extra-style footnotes (disabled by default; you enable them with the flag `MKD_EXTRA_FOOTNOTES`.)

- version 2.0.5 cleans up a long-standing compile-time warning, lightly tweaks the table of contents generator (`-ftoc`) to use `<a name=` anchors instead of `<hXid=` anchors, tweaks the `librarian.sh` generator so it doesn't require symbolic links to generate static libraries, and expands the characters allowed in css class names to include dashes and underscores.

- version 2.0.4 corrects a few bugs, adds some chrome, and adds a few features.

  1. add shared library support (`--shared` in `configure.sh`; FreeBSD, Linux, and MacOSX only for now.)
  2. correct some typos
  3. Add manpages for the example programs `makepage(1)` and `mkd2html(1)`
  4. Initialize the random number generator and tag list separately, just in case someone really wants to do multiple initialize/deallocate cycles.
  5. Add a library destructor function (mkd_shlib_destructor) to discard the kw array before unloading a shared library.
  6. Put in a bunch of useless code to make gcc STFU.
  7. Be more picky about what comes before a ^ if we're superscripting
  8. Unclosed html/style blocks revert to being regular old markup. True, the markup will contain the opening tag (and most of the other html from that block) but it will be regular old markup with additional garbage added in.
  9. Null-terminate the buffers generated by `mkd_line()`, `mkd_css()` and `mkd_toc()`.
  10. Flip the order of arguments to `mkd_generatecss()`; should be `1,size`, not `size,1`.
  11. It turns out that non-toplevel paragraphs consume adjacent code blocks :-(
  12. Rework (again) indentation inside list items. Indent should never be > 4 for enumerated lists; this means that I have to test and cut back large indents and change my test cases to account for leading whitespace on deeply (2 spaces + enumerated marker + 1 space) indented items.
  13. Add a raft of testcases to cover the new features + bugfixes.
  14. Add markdown extra style definition list support
      - Add a configuration flag (`--with-dl=`{**discount|extra|both**} to select which way to do definition lists. Put a `DL=` into version.c so the user can see how this is configured.
      - If dl is turned off at compile time, mark it off in version.c

- version 2.0.3 updates the woefully out of date documentation describing the various `MKD_` flags available in the published interface. As a little bit of chrome, it also modified `mkd_document()` to put a trailing null at the end of the buffer containing the generated document.

- version 2.0.2 fixes a couple of defects, adds more testscripts, and adds and updates a couple of features:

1. Fix a glitch in new-style `[]` processing that would collapse two sequential new-style `[link]`s into one no matter how much space was between them. The reference dingus only coalesces adjacent new-style `[link]`s together if they are physically adjacent or have one space between them.
2. Modify superscript grabbing so that it grabs parenthetical and alphanumeric blocks
3. Rework (again!) the guts of `linkylinky` to support markdown 1.0.2b8-style `<..>` links: these are like the original format I implemented before I rolled them out while chasing compatability in that a url starting with '<' contains *all characters* until an unescaped '>' is encountered.
4. Change the autolink prefix to closer fit to the reference dingus; if a `<url>` has a known protocol prefix, it does not need a leading `//` after the prefix.
5. Add a couple of functions that write the current runtime configuration (either a flags value (`mkd_flags_are`) or of an MMIOT (`mkd_mmiot_flags`)) to a FILE. `markdown -VV` prints out the flags for now, and the functions are there for other people to use as they desire (see below.)
6. In `theme`, add the pseudo-tag `<?theme config?>`, which writes a listing of all of the runtime settings to the output file. If `<?theme config?>` is called from the header section of `page.theme`, it writes the runtime settings in `markdown -VV` format, otherwise it writes a two-column html table (regrettably this will not work from inside generated content; theme is kind of stupid that way.)
7. Add test cases for <> links (regular and markdown 1 compat), split the superscript tests out from smarty.t

- version 2.0.1 fixes a newly introduced bug in _ handling (I flaked out and reversed the test for relaxed emphasis, but hadn't updated the testcase to account for the new runtime configuration for relaxed emphasis,) plus adds a pair of test cases for table of contents support.

- version 2.0 fixes one bug in table-of-contents handling (it would dump core if a header item contained a link), fixes the automated labeller so that it creates valid xhtml `id` attributes, and (by the simple expedient of breaking the published interface by expanding the flags field from an int to an at least 32 bit scalar (to fit the 22-and-counting option flags that discount now has)) restructures the code so that the only things that are compile-time options are memory allocation debugging and proper tab settings.

  It's a `.0` release, of course, so there may be some horrible bugs just waiting to fall out in the places I've not written test cases for yet, but there are many advantages to having the vast majority of the features be settable at runtime and I decided that this was a good time to make the change.

Archived releases

older versions of the code are still available.

Trivia

1. This document is generated from markdown source.
2. I’ve got a mirror of my sccs repository at github.