MoinMoin Documentation

Release 0.1.dev131+g03a1f12

The MoinMoin developers

May 21, 2022

Contents

Warning: This documentation **only** applies to **MoinMoin version 2** (aka moin2, moin 2.0, mm2, MoinMoin2, etc.), except where explicitly noted otherwise. Moin2 is very different from moin 1.x, so docs from one version will not apply to the other.

CHAPTER 1

General

1.1 About MoinMoin

MoinMoin is a wiki engine written in Python. It is Free and Open Source Software under GNU GPL v2+. For details please read the *License*.

Project homepage: https://moinmo.in/

Using MoinMoin, wiki users can easily create and maintain web content from their browser.

You can use it:

- as an easily-maintained web site
- as a knowledge base
- for taking notes
- for creating documentation

You can use it for:

- your company / organisation, your work group
- your school, college, or university
- your projects and interests
- just yourself

You can run it on:

- a public web server
- an intranet server
- your desktop or laptop
- Linux, Mac OS X, Windows, and other OSes

1.2 What makes MoinMoin special?

Moin tries to be a **great wiki engine**, which encompasses: powerful, extendable and easy-to-use. We don't try to be everything, but we don't try to be minimalistic either.

There are lots of wiki engines out there, making it hard to pick one. However, choosing wisely is important because you may have to live with your choice for a long time because switching wiki engines is not easy.

We won't list all of moin's features, because comparing feature lists is just not enough. Some features are best left unimplemented, even if they sound great at first. In moin, you will find most important features like in most major wiki engines. But still, you and your wiki users might feel quite a different overall experience just because of a bunch of small, superficial differences. Of course the quality of some features' implementations can vary greatly. Thus, you have to try it and play with it, not just look at feature comparisons.

MoinMoin has **been around since about 2000**. It has rapidly grown and evolved through moin 1.9.x. Its developers have increased their experience with Python and wiki technology over the years. With **moin 2.0**, there has been a rather **revolutionary cleanup / rewrite** of how moin works based on that experience. This promises to make it easier, cleaner, more consistent, more powerful, more flexible and more modular.

Moin is **written in Python**, an easy to read, high-level, object-oriented, dynamic, well-designed and platform-independent programming language.

Moin is **Free Software** (that implies that it is **Open Source**) and, because we use Python, you may even *like* to read and modify moin's code.

1.3 Who is using MoinMoin?

This shows some of the better-known users of MoinMoin:

1.3.1 Web Sites

- KernelNewbies, Xen, LinuxWireless, GCC
- Debian, Ubuntu, CentOS
- Apache, Gnome, Wine, OpenOffice, Squid, Exim, Dovecot
- Python, ScyPy, TurboGears
- Mercurial, Darcs
- FSFE, FFII, c-base, MusicBrainz
- linuxwiki.de, jurawiki.de, ooowiki.de and ... moinmo.in :D

For links and more sites, please see: https://moinmo.in/MoinMoinWikis

You may also add missing moin-based sites there.

1.3.2 Intranet installations

We know that there are a lot of private intranet installations of MoinMoin in:

- enterprises, companies
- government and administration
- scientific research facilities, universities, schools

- communities

Unfortunately, we do not have permission to name them here.

CHAPTER 2

Features

2.1 Operating System Support

Moin is implemented in Python, a platform-independent language. It works on Linux, Mac OS X, Windows, FreeBSD and other OSes that support Python.

That said, Linux is the preferred and most tested deployment platform and will likely have fewer issues than, for example, Windows.

2.2 Servers

- Builtin Python server from werkzeug, which is easy to use.
- Any server that talks WSGI to moin:
  - Apache2 with mod_wsgi
  - nginx with uwsgi
  - IIS with isapi-wsgi (not recommended - if you must use Windows, but have a choice concerning the web server, please use Apache2).
  - Other WSGI servers, see http://wsgi.org/
- With the help of flup middleware about any other server speaking:
  - fastcgi
  - scgi
  - ajp
  - cgi (slow, not recommended)

2.3 Authentication

- Builtin - username / password login form of moin, MoinAuth
- Builtin HTTP Basic Auth - browser login form, HTTPAuthMoin
- Auth against LDAP / Active Directory (LDAPAuth)
- Any authentication your web server supports via GivenAuth

2.4 Authorization

- Content Access Control Lists (ACLs)
  - global, using a mapping, so you can apply ACLs on parts of the namespace
  - local, per wiki item
  - give rights, such as:
    - create, destroy
    - read, write, rename
    - admin
  - to:
    - specific users
    - specific groups of users
    - all logged-in users
    - all users
- Function ACLs

2.5 Anti-Spam

- Form Ticketing

2.6 Storage

2.6.1 Item Types

- we store data of any type, such as text, images, audio, binary
- we separately store any metadata
- everything is revisioned

2.6.2 Storage Backend Types

- file system
- sqlite3
- everything supported by SQLalchemy
- you can easily add your own backend with little code

2.6.3 Serialization

- dump backend contents to a single file
- load backend contents from such a file

2.7 Search / Indexing

- important metadata is indexed
- content data is converted (if possible) and indexed
- fast indexed search, fast internal operations
- flexible and powerful search queries
- search current and historical contents
- using a shared index, find content in any farm wiki

2.8 User Interface

2.8.1 OO user interface

- Most functionality is done in the same way no matter what type your wiki item has.

2.8.2 Templating

- Theme support / User interface implemented with templates

2.8.3 Wiki features

- Global History for all items (full list)
- Latest Changes (“Recent Changes”), only lists the latest changes of an item
- Local History for one item (“History”)
- Diffs between any revision
  - text item diffs, rendered nicely with html
  - image diffs
  - binary “diff” (same or not same)

- Tags / Tag Cloud
- Missing Items
- Orphaned Items
- “What refers here?” functionality
- “What did I contribute to?” functionality
- Sitemap
- Macro support
- Multiple names and Namespaces support

2.8.4 Markup support

- Moin Wiki
- Creole
- MediaWiki
- reST
- DocBook XML
- Markdown
- HTML
- plus code / text file highlighting for many formats

2.8.5 Feeds

- Atom
- Google Sitemap

2.8.6 Notification

- by email: smtp or sendmail

2.8.7 Translation / Localization

- currently English and German translations only; no others will be added until the code and texts for moin2 are more stable
- any localization, provided by babel / pytz

2.9 Logging

- Flexible logging provided by *logging* module of python stdlib

2.10 Technologies

- html5, css, javascript with jquery, svg
- python
- flask, flask-caching, flask-babel, flask-theme, flask-script
- whoosh, werkzeug, pygments, flatland, blinker, babel, emeraldtree
- sqlalchemy (supports all popular SQL DBMS), sqlite, kyoto tycoon/cabinet

CHAPTER 3

License

```
MoinMoin's Copyright and License
================================

Copyright (c) 2000-2006 by Juergen Hermann <jh@web.de>
Copyright (c) 2006-2012 The MoinMoin development team, see
                        http://moinmo.in/MoinCoreTeamGroup

MoinMoin is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 2 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program.  If not, see <http://www.gnu.org/licenses/>.

See docs/licenses/COPYING for details.
```

For a FAQ about the GPL and a copy of the misc. GPL license versions, please see there: http://www.gnu.org/licenses/gpl.html

This is the GNU GPL version 2. From file docs/licenses/COPYING:

```
                    GNU GENERAL PUBLIC LICENSE
                       Version 2, June 1991

 Copyright (C) 1989, 1991 Free Software Foundation, Inc.,
 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA
 Everyone is permitted to copy and distribute verbatim copies
```

(continues on next page)

(continued from previous page)

```
 of this license document, but changing it is not allowed.

                            Preamble

  The licenses for most software are designed to take away your
freedom to share and change it.  By contrast, the GNU General Public
License is intended to guarantee your freedom to share and change free
software--to make sure the software is free for all its users.  This
General Public License applies to most of the Free Software
Foundation's software and to any other program whose authors commit to
using it.  (Some other Free Software Foundation software is covered by
the GNU Lesser General Public License instead.)  You can apply it to
your programs, too.

  When we speak of free software, we are referring to freedom, not
price.  Our General Public Licenses are designed to make sure that you
have the freedom to distribute copies of free software (and charge for
this service if you wish), that you receive source code or can get it
if you want it, that you can change the software or use pieces of it
in new free programs; and that you know you can do these things.

  To protect your rights, we need to make restrictions that forbid
anyone to deny you these rights or to ask you to surrender the rights.
These restrictions translate to certain responsibilities for you if you
distribute copies of the software, or if you modify it.

  For example, if you distribute copies of such a program, whether
gratis or for a fee, you must give the recipients all the rights that
you have.  You must make sure that they, too, receive or can get the
source code.  And you must show them these terms so they know their
rights.

  We protect your rights with two steps: (1) copyright the software, and
(2) offer you this license which gives you legal permission to copy,
distribute and/or modify the software.

  Also, for each author's protection and ours, we want to make certain
that everyone understands that there is no warranty for this free
software.  If the software is modified by someone else and passed on, we
want its recipients to know that what they have is not the original, so
that any problems introduced by others will not reflect on the original
authors' reputations.

  Finally, any free program is threatened constantly by software
patents.  We wish to avoid the danger that redistributors of a free
program will individually obtain patent licenses, in effect making the
program proprietary.  To prevent this, we have made it clear that any
patent must be licensed for everyone's free use or not licensed at all.

  The precise terms and conditions for copying, distribution and
modification follow.

                    GNU GENERAL PUBLIC LICENSE
   TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

  0. This License applies to any program or other work which contains
a notice placed by the copyright holder saying it may be distributed
```

(continues on next page)

(continued from previous page)

```
under the terms of this General Public License.  The "Program", below,
refers to any such program or work, and a "work based on the Program"
means either the Program or any derivative work under copyright law:
that is to say, a work containing the Program or a portion of it,
either verbatim or with modifications and/or translated into another
language.  (Hereinafter, translation is included without limitation in
the term "modification".)  Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not
covered by this License; they are outside its scope.  The act of
running the Program is not restricted, and the output from the Program
is covered only if its contents constitute a work based on the
Program (independent of having been made by running the Program).
Whether that is true depends on what the Program does.

  1. You may copy and distribute verbatim copies of the Program's
source code as you receive it, in any medium, provided that you
conspicuously and appropriately publish on each copy an appropriate
copyright notice and disclaimer of warranty; keep intact all the
notices that refer to this License and to the absence of any warranty;
and give any other recipients of the Program a copy of this License
along with the Program.

You may charge a fee for the physical act of transferring a copy, and
you may at your option offer warranty protection in exchange for a fee.

  2. You may modify your copy or copies of the Program or any portion
of it, thus forming a work based on the Program, and copy and
distribute such modifications or work under the terms of Section 1
above, provided that you also meet all of these conditions:

    a) You must cause the modified files to carry prominent notices
    stating that you changed the files and the date of any change.

    b) You must cause any work that you distribute or publish, that in
    whole or in part contains or is derived from the Program or any
    part thereof, to be licensed as a whole at no charge to all third
    parties under the terms of this License.

    c) If the modified program normally reads commands interactively
    when run, you must cause it, when started running for such
    interactive use in the most ordinary way, to print or display an
    announcement including an appropriate copyright notice and a
    notice that there is no warranty (or else, saying that you provide
    a warranty) and that users may redistribute the program under
    these conditions, and telling the user how to view a copy of this
    License.  (Exception: if the Program itself is interactive but
    does not normally print such an announcement, your work based on
    the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole.  If
identifiable sections of that work are not derived from the Program,
and can be reasonably considered independent and separate works in
themselves, then this License, and its terms, do not apply to those
sections when you distribute them as separate works.  But when you
distribute the same sections as part of a whole which is a work based
on the Program, the distribution of the whole must be on the terms of
```

(continues on next page)

(continued from previous page)

```
this License, whose permissions for other licensees extend to the
entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest
your rights to work written entirely by you; rather, the intent is to
exercise the right to control the distribution of derivative or
collective works based on the Program.

In addition, mere aggregation of another work not based on the Program
with the Program (or with a work based on the Program) on a volume of
a storage or distribution medium does not bring the other work under
the scope of this License.

  3. You may copy and distribute the Program (or a work based on it,
under Section 2) in object code or executable form under the terms of
Sections 1 and 2 above provided that you also do one of the following:

    a) Accompany it with the complete corresponding machine-readable
    source code, which must be distributed under the terms of Sections
    1 and 2 above on a medium customarily used for software interchange; or,

    b) Accompany it with a written offer, valid for at least three
    years, to give any third party, for a charge no more than your
    cost of physically performing source distribution, a complete
    machine-readable copy of the corresponding source code, to be
    distributed under the terms of Sections 1 and 2 above on a medium
    customarily used for software interchange; or,

    c) Accompany it with the information you received as to the offer
    to distribute corresponding source code.  (This alternative is
    allowed only for noncommercial distribution and only if you
    received the program in object code or executable form with such
    an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for
making modifications to it.  For an executable work, complete source
code means all the source code for all modules it contains, plus any
associated interface definition files, plus the scripts used to
control compilation and installation of the executable.  However, as a
special exception, the source code distributed need not include
anything that is normally distributed (in either source or binary
form) with the major components (compiler, kernel, and so on) of the
operating system on which the executable runs, unless that component
itself accompanies the executable.

If distribution of executable or object code is made by offering
access to copy from a designated place, then offering equivalent
access to copy the source code from the same place counts as
distribution of the source code, even though third parties are not
compelled to copy the source along with the object code.

  4. You may not copy, modify, sublicense, or distribute the Program
except as expressly provided under this License.  Any attempt
otherwise to copy, modify, sublicense or distribute the Program is
void, and will automatically terminate your rights under this License.
However, parties who have received copies, or rights, from you under
this License will not have their licenses terminated so long as such
```

(continues on next page)

(continued from previous page)

```
parties remain in full compliance.

  5. You are not required to accept this License, since you have not
signed it.  However, nothing else grants you permission to modify or
distribute the Program or its derivative works.  These actions are
prohibited by law if you do not accept this License.  Therefore, by
modifying or distributing the Program (or any work based on the
Program), you indicate your acceptance of this License to do so, and
all its terms and conditions for copying, distributing or modifying
the Program or works based on it.

  6. Each time you redistribute the Program (or any work based on the
Program), the recipient automatically receives a license from the
original licensor to copy, distribute or modify the Program subject to
these terms and conditions.  You may not impose any further
restrictions on the recipients' exercise of the rights granted herein.
You are not responsible for enforcing compliance by third parties to
this License.

  7. If, as a consequence of a court judgment or allegation of patent
infringement or for any other reason (not limited to patent issues),
conditions are imposed on you (whether by court order, agreement or
otherwise) that contradict the conditions of this License, they do not
excuse you from the conditions of this License.  If you cannot
distribute so as to satisfy simultaneously your obligations under this
License and any other pertinent obligations, then as a consequence you
may not distribute the Program at all.  For example, if a patent
license would not permit royalty-free redistribution of the Program by
all those who receive copies directly or indirectly through you, then
the only way you could satisfy both it and this License would be to
refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under
any particular circumstance, the balance of the section is intended to
apply and the section as a whole is intended to apply in other
circumstances.

It is not the purpose of this section to induce you to infringe any
patents or other property right claims or to contest validity of any
such claims; this section has the sole purpose of protecting the
integrity of the free software distribution system, which is
implemented by public license practices.  Many people have made
generous contributions to the wide range of software distributed
through that system in reliance on consistent application of that
system; it is up to the author/donor to decide if he or she is willing
to distribute software through any other system and a licensee cannot
impose that choice.

This section is intended to make thoroughly clear what is believed to
be a consequence of the rest of this License.

  8. If the distribution and/or use of the Program is restricted in
certain countries either by patents or by copyrighted interfaces, the
original copyright holder who places the Program under this License
may add an explicit geographical distribution limitation excluding
those countries, so that distribution is permitted only in or among
countries not thus excluded.  In such case, this License incorporates
```

(continues on next page)

(continued from previous page)

```
the limitation as if written in the body of this License.

  9. The Free Software Foundation may publish revised and/or new versions
of the General Public License from time to time.  Such new versions will
be similar in spirit to the present version, but may differ in detail to
address new problems or concerns.

Each version is given a distinguishing version number.  If the Program
specifies a version number of this License which applies to it and "any
later version", you have the option of following the terms and conditions
either of that version or of any later version published by the Free
Software Foundation.  If the Program does not specify a version number of
this License, you may choose any version ever published by the Free Software
Foundation.

  10. If you wish to incorporate parts of the Program into other free
programs whose distribution conditions are different, write to the author
to ask for permission.  For software which is copyrighted by the Free
Software Foundation, write to the Free Software Foundation; we sometimes
make exceptions for this.  Our decision will be guided by the two goals
of preserving the free status of all derivatives of our free software and
of promoting the sharing and reuse of software generally.

                            NO WARRANTY

  11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY
FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW.  EXCEPT WHEN
OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES
PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED
OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.  THE ENTIRE RISK AS
TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU.  SHOULD THE
PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING,
REPAIR OR CORRECTION.

  12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING
WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR
REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES,
INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING
OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED
TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY
YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER
PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE
POSSIBILITY OF SUCH DAMAGES.

                     END OF TERMS AND CONDITIONS

            How to Apply These Terms to Your New Programs

  If you develop a new program, and you want it to be of the greatest
possible use to the public, the best way to achieve this is to make it
free software which everyone can redistribute and change under these terms.

  To do so, attach the following notices to the program.  It is safest
to attach them to the start of each source file to most effectively
convey the exclusion of warranty; and each file should have at least
the "copyright" line and a pointer to where the full notice is found.
```

(continues on next page)

(continued from previous page)

```
    <one line to give the program's name and a brief idea of what it does.>
    Copyright (C) <year>  <name of author>

    This program is free software; you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation; either version 2 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License along
    with this program; if not, write to the Free Software Foundation, Inc.,
    51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this
when it starts in an interactive mode:

    Gnomovision version 69, Copyright (C) year name of author
    Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
    This is free software, and you are welcome to redistribute it
    under certain conditions; type `show c' for details.

The hypothetical commands `show w' and `show c' should show the appropriate
parts of the General Public License.  Of course, the commands you use may
be called something other than `show w' and `show c'; they could even be
mouse-clicks or menu items--whatever suits your program.

You should also get your employer (if you work as a programmer) or your
school, if any, to sign a "copyright disclaimer" for the program, if
necessary.  Here is a sample; alter the names:

  Yoyodyne, Inc., hereby disclaims all copyright interest in the program
  `Gnomovision' (which makes passes at compilers) written by James Hacker.

  <signature of Ty Coon>, 1 April 1989
  Ty Coon, President of Vice

This General Public License does not permit incorporating your program into
proprietary programs.  If your program is a subroutine library, you may
consider it more useful to permit linking proprietary applications with the
library.  If this is what you want to do, use the GNU Lesser General
Public License instead of this License.
```

CHAPTER 4

Using MoinMoin

4.1 User Accounts

Accounts provide an easy way for wiki users to identify themselves to MoinMoin and other wiki users, store personal preferences and track wiki contributions. Account creation is simple and straightforward, and provides many benefits over browsing and editing anonymously.

4.1.1 Account Creation

To create an account, click the *Login* button at the top of the page. You will be taken to a login page allowing you to either log in or create an account. Proceed to the create account page by clicking the account creation button, and you will be presented with an account creation form. The fields of this form are as follows:

Name Your username on the wiki. Will appear in the history section of any wiki item which you edit. This is a required field.

Password Your password for logging into your new account. Remember to pick a strong password with a mix of upper and lower case letters, numbers and symbols. This is also a required field.

Password Enter your new password again (same as the above field). This is a required field to make sure that your first password entry was correct.

E-Mail The email address which will be associated with your new account. This can be used by the wiki administrators to contact you or to verify your account if email verification is enabled on the wiki. This is a required field.

Note: Some wikis require email verification, in which case you will have click an activation link which will be sent to the email address you provide. You must complete this step before you start using the wiki.

Other wikis may limit new account creation to administrators only to prevent the creation of bogus accounts and spamming bots. In this case you will have to contact the administrator to request an account.

4.1.2 User Settings

User settings provide a way for to customise your MoinMoin experience and perform account maintenance functions like changing email address or password. To access your settings page, click the *Settings* button at the top of the page.

The settings page appears as a list of links to various sub-pages for changing elements of your wiki experience, each of these sub-pages are listed below:

Personal Settings

Personal settings include wiki language and locale, username and alias.

Name Your username, as it will appear on the wiki and in the history pages of wiki items which you edit.

Display-Name The display name can be used to override your username, so you will still log in using your username but your display name will be displayed to other users and in your history page.

Timezone Setting this value will display edit times converted to your local time zone. For example, an edit time of 10AM UTC would appear as 8PM AEST if you changed your time zone to GMT +10/Australian Eastern Standard Time.

Locale Your preferred language for interacting with MoinMoin. Edit dates and times are formatted based upon the locale unless the ISO 8601 option is selected under Options.

Change Password

Password changes are recommended if you believe that the password you are using has been compromised.

Current Password Enter the password which you currently use to log into the wiki. This prevents passers-by from changing the password of a logged in account. This is a required field.

New Password The new password which you would like to use. This is a required field.

New Password (repeat) Enter your new password again. Used to detect typographical errors. This is a required field.

Notification Settings

Notification settings allow you to configure the way MoinMoin notifies you of changes and important information.

E-Mail Change the email address MoinMoin sends emails to.

Wiki Appearance Settings

Appearance settings allow you to customise the look and feel of the wiki.

Theme name The bundled MoinMoin wiki theme which you would like to use.

User CSS URL If you want to style MoinMoin with custom Cascading Style Sheets (CSS), enter a URL for your custom stylesheet here. Custom CSS provides an advanced level of control over appearance of MoinMoin pages.

Number rows in edit textarea The size (in lines) of MoinMoin's plain text editor when you edit an item. The default of 0 resizes the textarea to hold the entire document being edited.

History results per page The number of edits you will see when you look at the history of an item.

Quick Links

Quick links enable users to add frequently referenced pages to the Navigation links. In most cases, users will use the "Add Link" or "Remove Link" controls within Item Views to add or remove quick links to local wiki items. Several different types of links may be added:

- To manually add a link to a local wiki item, prefix the item name with the wiki name: MyWiki/myitem
- To add a link to an external wiki page, use the wiki name as a prefix: MeatBall/RecentChanges
- To add a link to an external web page, use the full URL: http://google.com
- Other types of links, such as mailto: may be added

Options

The "Options" section allows you to control privacy and advanced features of MoinMoin.

Always use ISO 8601 date-time format Display dates and times in ISO 8601 format rather than the usual Babel formats based upon the user's locale. If the UTC time zone is selected, dates and times will have a "z" suffix indicating the date or time is a UTC Zulu time.

Publish my email (not my wiki homepage) in author info Control whether or not other wiki users may see your email address.

Open editor on double click This option allows you to simply double click the text on any MoinMoin item and have it opened in the editor. When using the MoinMoin text editor, the textarea caret will be positioned on the paragraph that was clicked. If the textarea is larger than the display window, pressing the right-arrow key will scroll the page so the caret is visible near the bottom of the window.

Show comment sections Show the comment sections for wiki items you view.

Disable this account forever Tick this box if you want to disable your account. Your username or alias will still show in the history pages of items you have edited, but you will no longer be able to log in using your account.

4.1.3 Special Features for Users with Accounts

Your User Page

You user page is a wiki space in which you may share information about yourself with other users of that wiki. It can be accessed by clicking the button with your username on it at the top of the screen, and is edited like a normal wiki item.

"My Changes"

To view your modifications to a wiki, click on `User` in the navigation area, then on `My Changes`. This will show a list of revisions you have made to wiki items sorted by date-time.

The first column will usually show an icon with a link to a diff showing the changes made at that revision. If the item was deleted, the icon will have a link to a revert dialog. If the item has only one revision, the icon will indicate the content type.

The second column will show the item name, aliases, or item ID (if the item was deleted) at that revision with a link to a revision display.

The remaining columns with display timestamps, sizes, revision numbers, and comments.

Bookmarking

Some MoinMoin users spend a lot of time sifting through the global changes list (accessible via the *History* button at the top of every MoinMoin page) looking for unread changes. To help users remember which revisions they have read and which they have yet to read, MoinMoin provides bookmarks. If you have read revisions up until the 13th of January, for example, you would simply click the *Set bookmark* button next to the revisions from the 13th of January to hide all revisions from before that date. If you wish to examine those revisions again, navigate back to the global history page and click *Remove bookmark*.

Quicklinks

At the top of every MoinMoin page, there is a row of buttons for quick access to commonly used MoinMoin features like the global index, global history and homepage. Often, users need quick access to MoinMoin items without having to search for them each time - quicklinks allow you to access your favourite wiki items at the click of a button by placing links to them at the top of every page. To quicklink an item, click the *Add Link* button at the top or bottom of a MoinMoin item. To remove a quicklink, simply navigate back to the item and click the *Remove Link* button.

Quicklinks are associated with your account, so you will be able to access them from anywhere by simply logging into the wiki.

Item Trail

The item trail appears at the top of each page and lists previous items which you have visited. Users with accounts may view this trail wherever they log in, whereas anonymous users have a different trail on each computer that they visit.

Subscribing to Items

Subscribing to items allows you to be notified via email when changes are made. To subscribe, navigate to the item in question and click the *Subscribe* button at the top or bottom of the page. You will now receive an email each time a user modifies this item. To unsubscribe, navigate to the item again and click the *Unsubscribe* button at the top or bottom of the page.

4.1.4 Logging out

Logging out of your account can prevent account hijacking on untrusted or insecure computers, and is considered best practice for security. To log out, click the *Logout* button at the top of the page. You will be redirected to a page confirming that you have logged out successfully.

4.2 Markups Supported by MoinMoin

4.2.1 Moin Wiki markup overview

This document describes the features of the moinwiki markup language. Because this document was created using Restructured Text, which does not support some of the features available in moinwiki, the examples below may show both the markup and result as block or predefined code.

Features currently not working with moin's Wiki parser are marked with **MOINTODO**.

Table Of Contents

Table of contents:

```
<<TableOfContents()>>
```

Table of contents (up to 2nd level headings only):

```
<<TableOfContents(2)>>
```

Headings

Markup:

```
= Level 1 =
== Level 2 ==
=== Level 3 ===
==== Level 4 ====
===== Level 5 =====
====== Level 6 ======
```

Result:

Level 1

Intentionally not rendered as level 1 so as to not interfere with Sphinx's indexing

Level 2

Level 3

Level 4

Level 5

Level 6

Notes:

- Closing equals signs are compulsory.
- Also, whitespace between the first word of the heading and the opening equals sign will not be shown in the output (ie. leading whitespace is stripped).

Text formatting

The following is a table of inline markup that can be used to control text formatting in Moin.

Markup	Result
`'''Bold Text'''`	**Bold text**
`''Italic''`	*Italic*
`'''''Bold Italic'''''`	Bold Italic
`` `Monospace` ``	`Monospace`
`{{{Code}}}`	`Code`
`__Underline__`	Underline
`^Super^Script`	Super Script
`,,Sub,,Script`	Sub Script
`~-Smaller-~`	Smaller
`~+Larger+~`	Larger
`--(Stroke)--`	Stroke

Hyperlinks

Moin2 hyperlinks are enclosed within double brackets. There are three possible fields separated by "|" characters:

```
1. PageName, relative URL, fully qualified URL, or interwiki link
2. Text description or transcluded icon: [[ItemName|{{MyLogo.png}}]]
3. Parameters: target, title, download, class, and accesskey are supported
```

The special CSS class *redirect* may be used to immediately redirect the browser to an internal or external page. Once placed inside an item, that item cannot be viewed as redirection is immediate. To edit the item, type . . . /+modify/ItemName in the browsers address bar.

Examples with parameters are not shown below because the effect cannot be duplicated with reST markup. To open a link in a new tab or window with a mouseover title, do:

```
* [[ItemName|my favorite item|target=_blank,title="Go There!"]]
```

Internal Links

Internal links for namespaces work the same as an item in the default namespace with subitems. Links without a leading / or ../ refer to an item in the top level of the default namespace, even if the current item is not in the default namespace. Links with a leading / refer to a subitem of the current item. Links with a leading ../ refer to a sibling of the current item.

Markup	Result	Comments
`[[ItemName]]`	ItemName	Link to an item
`[[ItemName|Named Item]]`	Named Item	Named link to an internal item
`[[#AnchorName]]`	*#AnchorName*	Link to an anchor in the current item
`[[#AnchorName|AnchorName]]`	*AnchorName*	Link to a named anchor
`[[ItemName#AnchorName]]`	ItemName#AnchorName	Link to an anchor in an internal item
`[[ItemName#AnchorName|Named Item1]]`	Named Item1	Named link to an anchor in an internal item
`[[../SiblingItem]]`	../SiblingItem	Link to a sibling of the current item
`[[/SubItem]]`	/SubItem	Link to a sub-item of current item
`[[Home/ItemName]]`	Home/ItemName	Link to a subitem of Home item
`[[/filename.txt]]`	/filename.txt	Link to a sub-item called Filename.txt
`[[users/JoeDoe]]`	**'users/JoeDoe'_**	Link to a user's home item in user namespace
`[[AltItem||class="redirect"]]`	*AltItem is displayed immediately*	Type /+modify/<item> in address bar to edit

External Links

Markup	Result	Comments
`[[https://moinmo.in/]]`	https://moinmo.in/	External link
`[[https://moinmo.in/|MoinMoin Wiki]]`	MoinMoin Wiki	Named External link
`[[MeatBall:InterWiki]]`	MeatBall:InterWiki	Link to an item on an external Wiki
`[[MeatBall:InterWiki|InterWiki page on MeatBall]]`	InterWiki page on MeatBall	Named link to an item on an external Wiki
`[[mailto:user@example.com]]`	mailto:user@example.com	Mailto link

Images and Transclusions

Transclusion syntax is defined as follows:

```
{{<target>|<optional alternate text>|<optional parameters>}}

{{bird.jpg|rare yellow bird|class=center}}
```

- *<target>* is a relative or absolute URL pointing to an image, video, audio, or web page.
- *<optional alternate text>* has several potential uses:
  - Screen readers used by visually impaired users will speak the text.
  - The alternate text may be displayed by the browser if the URL is broken.
  - Search engine crawlers may use the text to index the page or image.
- *optional parameters* may be used to resize or position the target.
  - the browser will automatically resize the image to fit the enclosing container by specifying either class=resize or width=100% height=auto
  - Images or other targets can be resized on the client side by specifying an option of *width=nn* or *height=nn* where nn is the desired size in pixels.

- If Pillow is installed on the server, JPEG (or JPG) images can be resized on the server by specifying an option of *&w=nn* or *&h=nn* where nn is the desired size in pixels.
- Images embedded within text can be positioned relative to a line of text by using *class=bottom*, *class=top* or *class="middle"*.
- Images displayed as block elements my be floated left or right by using *class="left"* or *class=right* respectively, or centered by using *class=center*.

Markup	Comment
`text {{example.png}} text`	Embed example.png inline
`text {{example.png||class=top height=96}} text`	Embed example.png inline
`{{example.png||class=center}}`	example.png as centered image
`{{https://static.moinmo.in/logos/moinmoin.png}}`	example.png aligned left, not float
`{{ItemName}}`	Transclude (embed the contents of) ItemName
`{{/SubItem}}`	Transclude SubItem
`{{ example.jpg || class=resize }}`	browser will automatically resize image to fit the enclosing container
`{{ example.jpg || width=20, height=100 }}`	Resizes example.png by using HTML tag attributes
`{{ example.jpg || &w=20 }}`	Resizes example.png by using server- side compression, requires PIL
`{{ https://moinmo.in/ || width=800 }}`	Resizes the `object` which is embedded using HTML tags. Here markup like `&w=800` will not work.

Extra Info:

Markup like `{{ example.jpg || &w=20 }}`, simply adds `&w` to the `src` URL of the image, the Python Imaging Library (PIL) understands that it has to compress the image on the server side and render as shrinked to size `20`.

For markup like `{{ example.jpg || width=20, height=100 }}` we currently allow only the `width` and `height` (anything else is ignored) to be added as attributes in the HTML, however one can, add anything to the query URL using `&`, like `&w` in the example above.

Most browsers will display a large blank space when a web page using an https protocol is transcluded into a page using http protocol. Transcluding a png image using an https protocol into an http protocol page displays OK in all browsers.

Blockquotes and Indentations

Markup:

```
indented text
 text indented to the 2nd level
```

Result:

indented text text indented to the 2nd level

Lists

Warning:

- All Moin Wiki list syntax (including that for unordered lists, ordered lists and definition lists) requires a leading space before each item in the list.
- Unfortunately, reStructuredText does not allow leading whitespace in code samples, so the example markup here will not work if copied verbatim, and requires that each line of the list be indented by one space in order to be valid Moin Wiki markup.
- This is also an **reSTTODO**

Unordered Lists

Markup:

```
* item 1
* item 2 (preceding white space)
 * item 2.1
  * item 2.1.1
* item 3
 . item 3.1 (bulletless)
. item 4 (bulletless)
 * item 4.1
  . item 4.1.1 (bulletless)
```

Result:

- item 1
- item 2 (preceding white space)
  - item 2.1
  - item 2.1.1
- item 3
- item 3.1 (bulletless)
- item 4 (bulletless)
  - item 4.1
  - item 4.1.1 (bulletless)

Note:

- Moin markup allows a square, white and a bulletless item for unordered lists, these cannot be shown in reST documents.

Ordered Lists

With Numbers

Markup:

```
1. item 1
  1. item 1.1
  1. item 1.2
1. item 2
```

Result:

1. item 1
   1. item 1.1
   2. item 1.2
2. item 2

With Roman Numbers

Markup:

```
I. item 1
  i. item 1.1
  i. item 1.2
I. item 2
```

Result:

```
I. item 1

  i. item 1.1

  ii. item 1.2

II. item 2
```

With Letters

Markup:

```
A. item 1
  a. item 1.1
  a. item 1.2
A. item 2
```

Result:

A. item 1
   a. item 1.1
   b. item 1.2
B. item 2

Specify a Starting Point

When there is a need to start an ordered list at a specific number, use the format below. This works for ordered lists using letters and roman numerals.

Markup:

```
1.#11 eleven
1. twelve
   i.#11 roman numeral xi
1. thirteen

A.#11 letter K
A. letter J
```

Result:

```
11. eleven
12. twelve
   xi.roman numeral xi
13. thirteen

K. letter K
J. letter J
```

Definition Lists

Markup:

```
term:: definition
object::
:: description 1
:: description 2
```

Result:

term definition

object

description 1
description 2

Notes:

- reStructuredText does not support multiple definitions for a single term, so a line break has been forced to illustrate the appearance of several definitions.
- Using the prescribed Moin Wiki markup will, in fact, produce two separate definitions in MoinMoin (using separate `<dd>` tags).

Horizontal Rules

To create a horizontal rule, start a line with 4 or more hyphen (-) characters. Nine (or more) characters creates a line of maximum height.

Markup:

```
Text
----
Text
```

Result:

Text

Text

Tables

Moin wiki markup supports table headers and footers. To indicate the first row(s) of a table is a header, insert a line of 3 or more = characters. To indicate a footer, include a second line of = characters after the body of the table.

Markup:

```
||Head A ||Head B ||Head C ||
============================
||a       ||b       ||c       ||
||x       ||y       ||z       ||
```

Result:

Head A	Head B	Head C
a	b	c
x	y	z

Table Styling

To add styling to a table, enclose one or more parameters within angle brackets at the start of any table cell. Options for tables must be within first cell of first row. Options for rows must be within first cell of the row. Separate multiple options with a blank character.

Markup	Effect
<tableclass="zebra moin-sortable">	Adds one or more CSS classes to the table
<rowclass="orange">	Adds one or more CSS classes to the row
<class="green">	Adds one or more CSS classes to the cell
<tablestyle="color: red;">	Add CSS styling to table
<rowstyle="font-size: 140%; ">	Add CSS styling to row
<style="text-align: right;">	Add CSS styling to cell
<bgcolor="#ff0000">	Add CSS background color to cell
<rowbgcolor="#ff0000">	Add CSS background color to row
<tablebgcolor="#ff0000">	Add CSS background color to table
width	Add CSS width to cell
tablewidth	Add CSS width to table
id	Add HTML ID to cell
rowid	Add HTML ID to row
tableid	Add HTML ID to table
rowspan	Add HTML rowspan attribute to cell
colspan	Add HTML colspan attribute to cell
caption	Add HTML caption attribute to table
<80%>	Set cell width, setting one cell effects entire table column
<(>	Align cell contents left
<)>	Align cell contents right
<:>	Center cell contents
<|2>	Cell spans 2 rows (omit a cell in next row)
<-2>	Cell spans 2 columns (omit a cell in this row)
<#0000FF>	Change background color of a table cell
<rowspan="2">	Same as <|2> above
<colspan="2">	Same as <-2> above
– no content –	An empty cell has same effect as <-2> above
===	A line of 3+ "=" separates table header, body and footer

Table Styling Example

Markup:

```
||Head A||Head B||
===
||normal text||normal text||
||<|2>cell spanning 2 rows||cell in the 2nd column||
||cell in the 2nd column of the 2nd row||
||<rowstyle="font-weight: bold;" class="monospaced">monospaced text||bold text||

||<tableclass="no-borders">A||B||C||
||D||E||F||
```

Result:

Head A	Head B
normal text	normal text
cell spanning 2 rows	cell in the 2nd column
	cell in the 2nd column of the 2nd row
`monospaced text`	**bold text**

A B C
D E F

Verbatim Display

To show plain text preformatted code, just enclose the text in three or more curly braces.

Markup:

```
{{{
no indentation example
}}}

   {{{{
   {{{
   indentation; using 4 curly braces to show example with 3 curly braces
   }}}
   }}}}
```

Result:

```
no indentation example

   {{{
   indentation; using 4 curly braces to show example with 3 curly braces
   }}}
```

Parsers

Syntax Highlighting

Markup:

```
{{{#!highlight python
def hello():
   print "Hello World!"
}}}
```

Result:

```
def hello():
    print "Hello, world!"
```

creole, rst, markdown, docbook, and mediawiki

To add a small section of markup using another parser, follow the example below replacing “creole” with the target parser name. The moinwiki parser does not have the facility to place table headings in the first column, but the creole parser can be used to create the desired table.

Markup:

```
{{{#!creole
|=X|1
|=Y|123
|=Z|12345
}}}
```

Result:

X	1
Y	123
Z	12345

csv

The default separator for CSV cells is a semi-colon (;). The example below specifies a comma (,) is to be used as the separator.

Markup:

```
{{{#!csv ,
Fruit,Color,Quantity
apple,red,5
banana,yellow,23
grape,purple,126
}}}
```

Result:

Fruit	Color	Quantity
apple	red	5
banana	yellow	23
grape	purple	126

wiki

The wiki parser is the moinwiki parser. If there is a need to emphasize a section, pass some predefined classes to the wiki parser.

Markup:

```
{{{#!wiki solid/orange
* plain
* ''italic''
* '''bold'''
* '''''bold italic.'''''
}}}
```

Result:

- plain
- ''italic''
- '''bold'''

- """bold italic."""

Admonitions

Admonitions are used to draw the reader's attention to an important paragraph. There are nine admonition types: attention, caution, danger, error, hint, important, note, tip, and warning.

Markup:

```
{{{#!wiki caution
'''Don't overuse admonitions'''

Admonitions should be used with care. A page riddled with admonitions
will look restless and will be harder to follow than a page where
admonitions are used sparingly.
}}}
```

Result:

> **Caution:** "'Don't overuse admonitions"'
>
> Admonitions should be used with care. A page riddled with admonitions will look restless and will be harder to follow than a page where admonitions are used sparingly.

CSS classes for use with the wiki parser, tables, comments, and links

- Background colors: red, green, blue, yellow, or orange
- Borders: solid, dashed, or dotted
- Text-alignment: left, center, right, or justify
- Admonitions: caution, important, note, tip, warning
- Tables: moin-sortable, no-borders
- Comments: comment
- Position parsers and tables: float-left, float-right, inline, middle, clear-right, clear-left or clear-both
- Links with browser redirection: redirect

Variables

Variables within the content of a moin wiki item are transformed when the item is saved. An exception is if the item has a tag of "'template"', then no variables are processed. This makes variables particularly useful within template items. Another frequent use is to add signatures (@SIG@) to a comment within a discussion item.

Variable expansion is global and happens everywhere within an item, including code displays, comments, tables, headings, inline parsers, etc.. Variables within transclusions are not expanded because they are not part of the including item's content.

TODO: Allow wiki admins and users to add custom variables. There is no difference between system date format and user date format in Moin 2, fix code or docs.

Predefined Variables

Variable	Description	Resulting Markup	Example Rendering
@PAGE@	Name of the item (useful for templates)	HelpOnPageCreation	HelpOnPageCreation
@ITEM@	Name of the item (useful for templates)	HelpOnPageCreation	HelpOnPageCreation
@TIMESTAMP@	Raw time stamp	2004-08-30T06:38:05Z	2004-08-30T06:38:05Z
@DATE@	Current date in the system format	<<Date(2004-08-30T06:38:05Z)>>	<<Date(2004-08-30T06:38:05Z)>>
@TIME@	Current date and time in the user format	<<DateTime(2004-08-30T06:38:05Z)>>	<<DateTime(2004-08-30T06:38:05Z)>>
@ME@	user's name or "anonymous"	TheAnarcat	TheAnarcat
@USERNAME@	user's name or his domain/IP	TheAnarcat	TheAnarcat
@USER@	Signature "– loginname"	– TheAnarcat	– TheAnarcat
@SIG@	Dated Signature "– login name date time"	– TheAnarcat <<DateTime(...)>>	– TheAnarcat <<DateTime(2004-08-30T06:38:05Z)>>
@EMAIL@	<<MailTo()>> macro, obfuscated email	<<MailTo(user AT example DOT com)>>	user@example.com OR user AT example DOT com
@MAILTO@	<<MailTo()>> macro	<<MailTo(testuser@example.com)>>	testuser@example.com, no obfuscation

Notes:

- @PAGE@ and @ITEM@ results are identical, item being a moin 2 term and page a moin 1.x term.
- If an editor is not logged in, then any @EMAIL@ or @MAILTO@ variables in the content are made harmless by inserting a space character. This prevents a subsequent logged in editor from adding his email address to the item accidentally.

Macros

Macros are extensions to standard markup that allow developers to add extra features. The following is a table of MoinMoin's macros.

Markup	Comment
`<<Anchor(anchorname)>>`	Inserts an anchor named "anchorname"
`< >`	Inserts a forced linebreak
`<<Date()>>`	Inserts current date, or unix timestamp or ISO 8601 date
`<<DateTime()>>`	Inserts current datetime, or unix timestamp or ISO 8601
`<<GetText(Settings)>>`	Loads I18N texts, Einstellungen if browser is set to German
`<<GetVal(WikiDict, var1)>>`	Loads var1 value from metadata of item named WikiDict
`<<FootNote(Note here)>>`	Inserts a footnote saying "Note here"
`<<FontAwesome(name, color,size)>>`	displays Font Awsome icon, color and size are optional
`<<Icon(my-icon.png)>>`	displays icon from /static/img/icons
`<<Include(ItemOne/SubItem)>>`	Embeds the contents of `ItemOne/SubItem` inline
`<<ItemList()>>`	Lists subitems of current item, see notes for options
`<<MailTo(user AT example DOT org, write me)>>`	If the user is logged in this macro will display `user@example.org`, otherwise it will display the obfuscated email address supplied (`user AT example DOT org`) The second parameter containing link text is optional.
`<<MonthCalendar()>>`	Shows a monthly calendar in a table form, see notes for details
`<<RandomItem(3)>>`	Inserts names of 3 random items
`<<ShowIcons()>>`	displays all icons in /static/img/icons directory
`<<TableOfContents(2)>>`	Shows a table of contents up to level 2
`<<Verbatim(`same` __text__)>>`	Inserts text as entered, no markup rendering

Notes

Date and **DateTime** macros accept integer timestamps and ISO 8601 formatted date-times:

- <<Date(1434563755)>>
- <<Date(2002-01-23T12:34:56)>>

Footnotes are created by placing the macro within text. By default footnotes are placed at the bottom of the page. Explicit placement of footnotes is accomplished by calling the macro without a parameter.

- text<<FootNote(A macro is enclosed in double angle brackets, and"'may"' have markup.)>> more text
- <<FootNote()>>

FontAwesome color must be a hex digit color code of either 3 or 6 digits with a leading #: #f00 or #F80000. FontAwesome size must be an unsigned decimal integer or float that will adjust the size of the character relative to the current font size: 2 or 2.0 will create double the character size, .5 will create a character half the current size. Font awesome experts will know about the special "fa" class and the "fa-" name prefixes. It is acceptable, but not necessary to provide these. See https://fontawesome.com/v4/cheatsheet/

- <<FontAwesome(thumbs-up,#f00,2)>> is identical to
- <<FontAwesome(fa fa-thumbs-up fa-2x,#FF0000)>>

The **Include** macro <<Include(my.png)>> produces results identical to the transclusion {{my.png}}. It is more flexible than a transclusion because it supports multiple parameters and the first parameter may be any regrex starting with a ^. The include macro accepts 3 parameters where the second parameter is a heading and the third parameter a heading level between 1 and 6:

- <<Include(^zi)>> embeds all wiki items starting with *zi*.
- <<Include(moin.png,My Favorite icon, 6)>>

The **ItemList** macro accepts multiple named parameters: item, startswith, regex, ordered and display.

- <<ItemList(item="Foo")>> lists subitems of Foo item
- <<ItemList(ordered='True')>> displays ordered list of subitems, default is unordered
- <<ItemList(startswith="Foo")>> lists subitems starting with Foo
- <<ItemList(regex="Foo$")>> lists subitems ending with Foo
- <<ItemList(skiptag="template")>> ignore items with this tag
- <<ItemList(display="FullPath")>> default, displays full path to subitems
- <<ItemList(display="ChildPath")>> displays last component of the FullPath, including the '/'
- <<ItemList(display="ChildName")>> displays subitem name
- <<ItemList(display="UnCameled")>> displays "fooBar" as "foo Bar"

The **MonthCalendar** macro accepts multiple named parameters: item, year, month, month_offset, fixed_height and anniversary.

- <<MonthCalendar>> Calendar of current month for current page
- <<MonthCalendar(month_offset=-1)>> Calendar of last month
- <<MonthCalendar(month_offset=+1)>> Calendar of next month
- <<MonthCalendar(item="SampleUser",month=12)>> Calendar of Page SampleUser, this year's december
- <<MonthCalendar(month=12)>> Calendar of current Page, this year's december
- <<MonthCalendar(year=2022,month=12)>> Calendar of December, 2022

Smileys and Icons

This table shows moin smiley markup, the rendering of smiley icons cannot be shown in Rest markup.

X-(	:D	<:(	:o
:(	:)	B)	:))
;)	/!\	<!>	(!)
:-?	:\	>:>	|)
:-(	:-)	B-)	:-))
;-)	|-)	(./)	{OK}
{X}	{i}	{1}	{2}
{3}	{*}	{o}	

Comments

There are three ways to add comments to a page. Lines starting with ## can be seen only by page editors. Phrases enclosed in /* and */ and wiki parser section blocks of text with a class of "comment" may be hidden or visible depending upon user settings or actions.

Markup:

```
## Lines starting with "##" may be used to give instructions
## to future page editors.

Click on the "Comments" button within Item Views to toggle the /* comments */␣
↪visibility.

{{{#!wiki comment/dashed
This is a wiki parser section with class "comment dashed".

Its visibility gets toggled by clicking on the comments button.
}}}
```

Result:

Click on the "Comments" button within Item Views to toggle the visibility.

Notes:

- The toggle display feature does not work on reST documents, so there is no way to see the hidden comments.

4.2.2 WikiCreole markup overview

Features currently not working with moin's WikiCreole parser are marked with **CREOLETODO**.

Features currently not working with moin's rst parser are marked with **reSTTODO**.

Headings

Markup:

```
= Level 1
== Level 2
=== Level 3
==== Level 4
===== Level 5
====== Level 6
```

Result:

Level 1

Intentionally not rendered as level 1 so it does not interfere with Sphinx's indexing

Level 2

Level 3

Level 4

Level 5

Level 6

Notes:

Closing equals signs are optional and do not affect the output. Also, whitespace between the first word of the heading and the opening equals sign will not be shown in the output (ie. leading whitespace is stripped).

Text formatting

The following is a table of inline markup that can be used to format text in Creole.

Markup	Result
`**Bold Text**`	**Bold text**
`//Italic Text//`	*Italic Text*
`//**Bold and Italic**//`	Bold and Italic
`__Underline__`	Underline
`{{{Monospace}}}`	`Monospace`
`First line\\Second line`	First line Second line

reSTTODO: Restructured Text line blocks are not working in Moin2

Hyperlinks

Internal links

Markup	Result	Comment
`[[ItemName]]`	*Item name*	Link to an item
`[[ItemName|Named Item]]`	*Named Item*	Named link to an internal item
`[[#AnchorName]]`	*#AnchorName*	Link to an anchor in the current item
`[[#AnchorName|Named anchor]]`	*Named anchor*	Link to a named anchor.
`[[ItemName#AnchorName]]`	*ItemName#AnchorName*	Link to an anchor in an internal item
`[[ItemName/SubItem]]`	*ItemName/Subitem*	Link to a sub-item of an internal item
`[[../SiblingItem]]`	*../SiblingItem*	Link to a sibling of the current item
`[[/SubItem]]`	*/SubItem*	Link to a sub-item
`[[attachment:Filename.txt]]`	*Filename.txt*	Link to a sub-item called Filename.txt. Note that this is for MoinMoin 1.x compatability and is deprecated in favour of the more convenient `[[/SubItem]]` syntax

External links

Markup	Result	Comment
`http://www.example.com`	http://www.example.com	External link
`[[http://www.example.com]]`	http://www.example.com	External link
`[[MeatBall:InterWiki|InterWiki item on MeatBall]]`	InterWiki item on MeatBall	Link to an item on an external Wiki
`[[mailto:user@example.org]]`	mailto:user@example.org	Mailto link

Images and Transclusions

Markup	Comment
`{{example.png}}`	Embed example.png inline
`{{example.png|Alt text}}`	Embed example.png inline or display "Alt text" if not available
`{{ItemName}}`	Transclude (embed the contents of) ItemName inline.
`{{/SubItem}}`	Transclude SubItem inline.

Paragraphs

Markup:

```
You can leave an empty line to start a new paragraph.

Single breaks are ignored.
To force a line break, use <<BR>> or \\.
```

Result:

You can leave an empty line to start a new paragraph.

Single breaks are ignored. To force a line break, use
or
.

reSTTODO: reStructuredText line blocks are not working in Moin2

Horizontal rules

Markup:

```
A horizontal rule can be added by typing four dashes.

----

This text will be displayed below the rule.
```

Result:

A horizontal rule can be added by typing four dashes.

This text will be displayed below the rule.

Preformatted text

Markup:

```
{{{
This text will [[escape]] **special** WikiCreole //markup//
    It will also preserve indents

And whitespace.
}}}
~[[This text will not be a link, because it uses the tilde (~) escape character]]
```

Result:

```
This text will [[escape]] **special** WikiCreole //markup//
    It will also preserve indents

And whitespace.
```

[[This text will not be a link, because it uses the tilde (~) escape character]]

Notes:

This tilde character (~) makes the parser ignore the character following it, which can be used to prevent links from appearing as links or prevent bold text from appearing as bold. For example "`~**Not bold~**`" would output "**Not bold**").

Syntax Highlighting

Markup:

```
{{{
#!python
#Python syntax highlighting
import this

def spam():
    print('Spam, glorious spam!')

spam()
}}}
```

Result:

```
#Python syntax highlighting
import this

def spam():
    print('Spam, glorious spam!')
```

(continues on next page)

(continued from previous page)

```
spam()
```

CREOLETODO:The use of syntax highlighting currently crashes moin.

Lists

Ordered lists

Ordered lists are formed of lines that start with number signs (#). The number of '#' signs at the beginning of a line determines the current level.

Markup:

```
# First item
# Second item
## First item (second level)
## Second item (second level)
### First item (third level)
# Third item
```

Result:

1. First item
2. Second item
    1. First item (second level)
    2. Second item (second level)
    1. First item (third level)
3. Third item

Unordered lists

Markup:

```
* List item
* List item
** List item (second level)
*** List item (third level)
* List item
```

Result:

- List item
- List item
    - List item (second level)
    - List item (third level)
- List item

Mixed lists

Markup:

```
# First item
# Second item
** Bullet point one
** Bullet point two
# Third item
# Fourth item
```

Result:

1. First item
2. Second item
   - Bullet point one
   - Bullet point two
3. Third item
4. Fourth item

Tables

Markup:

```
|= Header one |= Header two |
| Cell one    | Cell two
| Cell three  | Cell four   |
```

Result:

Header one	Header two
Cell one	Cell two
Cell three	Cell four

Notes:

Table cells start with a pipe symbol (`|`), and header cells start with a pipe symbol and equals sign (`|=`). The closing pipe symbol at the end of a row is optional.

Macros

Macros are extensions to standard Creole markup that allow developers to add extra features. The following is a table of MoinMoin's Creole macros.

Markup	Comment
`<<Anchor(anchorname)>>`	Inserts an anchor named "anchorname"
`< >`	Inserts a forced linebreak
`<<Date()>>`	Inserts current date, or unix timestamp or ISO 8601 date
`<<DateTime()>>`	Inserts current datetime, or unix timestamp or ISO 8601
`<<GetText(Settings)>>`	Loads I18N texts, Einstellungen if browser is set to German
`<<GetVal(WikiDict, var1)>>`	Loads var1 value from metadata of item named WikiDict
`<<FootNote(Note here)>>`	Inserts a footnote saying "Note here"
`<<FontAwesome(name, color,size)>>`	displays Font Awsome icon, color and size are optional
`<<Icon(my-icon.png)>>`	displays icon from /static/img/icons
`<<Include(ItemOne/SubItem)>>`	Embeds the contents of `ItemOne/SubItem` inline
`<<ItemList()>>`	Lists subitems of current item, see notes for options
`<<MailTo(user AT example DOT org, write me)>>`	If the user is logged in this macro will display `user@example.org`, otherwise it will display the obfuscated email address supplied (`user AT example DOT org`) The second parameter containing link text is optional.
`<<MonthCalendar()>>`	Shows a monthly calendar in a table form, see notes for details
`<<RandomItem(3)>>`	Inserts names of 3 random items
`<<ShowIcons()>>`	displays all icons in /static/img/icons directory
`<<TableOfContents(2)>>`	Shows a table of contents up to level 2
`<<Verbatim(`same` __text__)>>`	Inserts text as entered, no markup rendering

Notes

Date and **DateTime** macros accept integer timestamps and ISO 8601 formatted date-times:

- <<Date(1434563755)>>
- <<Date(2002-01-23T12:34:56)>>

Footnotes are created by placing the macro within text. By default footnotes are placed at the bottom of the page. Explicit placement of footnotes is accomplished by calling the macro without a parameter.

- text<<FootNote(A macro is enclosed in double angle brackets, and"'may"' have markup.)>> more text
- <<FootNote()>>

FontAwesome color must be a hex digit color code of either 3 or 6 digits with a leading #: #f00 or #F80000. FontAwesome size must be an unsigned decimal integer or float that will adjust the size of the character relative to the current font size: 2 or 2.0 will create double the character size, .5 will create a character half the current size. Font awesome experts will know about the special "fa" class and the "fa-" name prefixes. It is acceptable, but not necessary to provide these. See https://fontawesome.com/v4/cheatsheet/

- <<FontAwesome(thumbs-up,#f00,2)>> is identical to
- <<FontAwesome(fa fa-thumbs-up fa-2x,#FF0000)>>

The **Include** macro <<Include(my.png)>> produces results identical to the transclusion {{my.png}}. It is more flexible than a transclusion because it supports multiple parameters and the first parameter may be any regrex starting with a ^. The include macro accepts 3 parameters where the second parameter is a heading and the third parameter a heading level between 1 and 6:

- <<Include(^zi)>> embeds all wiki items starting with *zi*.
- <<Include(moin.png,My Favorite icon, 6)>>

The **ItemList** macro accepts multiple named parameters: item, startswith, regex, ordered and display.

- <<ItemList(item="Foo")>> lists subitems of Foo item
- <<ItemList(ordered='True')>> displays ordered list of subitems, default is unordered
- <<ItemList(startswith="Foo")>> lists subitems starting with Foo
- <<ItemList(regex="Foo$")>> lists subitems ending with Foo
- <<ItemList(skiptag="template")>> ignore items with this tag
- <<ItemList(display="FullPath")>> default, displays full path to subitems
- <<ItemList(display="ChildPath")>> displays last component of the FullPath, including the '/'
- <<ItemList(display="ChildName")>> displays subitem name
- <<ItemList(display="UnCameled")>> displays "fooBar" as "foo Bar"

The **MonthCalendar** macro accepts multiple named parameters: item, year, month, month_offset, fixed_height and anniversary.

- <<MonthCalendar>> Calendar of current month for current page
- <<MonthCalendar(month_offset=-1)>> Calendar of last month
- <<MonthCalendar(month_offset=+1)>> Calendar of next month
- <<MonthCalendar(item="SampleUser",month=12)>> Calendar of Page SampleUser, this year's december
- <<MonthCalendar(month=12)>> Calendar of current Page, this year's december
- <<MonthCalendar(year=2022,month=12)>> Calendar of December, 2022

4.2.3 reST (ReStructured Text) Markup

Depending upon your source, this document may have been created by the Moin2 reST parser (Docutils) or the Sphinx reST parser. These parsers have slight differences in the rendering of reST markup, some of those differences are noted below.

The purpose of this document is to define the features of the Moin2 reST (Docutils) parser. The Sphinx extensions to reST markup that are not supported by the Docutils parser are not included here.

See the the Docutils Restructured Text documentation for more information.

Headings

Rather than imposing a fixed number and order of section title adornment styles, the order enforced will be the order as encountered. The first style encountered will be an outermost title (like HTML H1), the second style will be a subtitle, the third will be a subsubtitle, and so on.

The underline below the title must at least be equal to the length of the title itself. Failure to comply results in messages on the server log. Skipping a heading (e.g. putting an H5 heading directly under an H3) results in a rendering error and an error message will be displayed instead of the expected page.

If any markup appears before the first heading on a page, then the first heading will be an H2 and all subsequent headings will be demoted by 1 level.

Markup:

```
=======
Level 1
=======

Level 2
=======

# levels 1 and 2 are not shown below, see top of page and this section heading.

Level 3
-------

Level 4
*******

Level 5
:::::::

Level 6
+++++++
```

Result:

Level 3

Level 4

Level 5

Level 6

Table of Contents

Markup:

```
.. contents::
```

Result:

Contents

The table of contents may appear above or floated to the right side due to CSS styling.

Text formatting

The following is a table of inline markup that can be used to format text in Moin.

Markup	Result
`**Bold Text**`	**Bold text**
`*Italic*`	*Italic*
``` ``Inline Literals`` ```	`Inline Literals`
`***nested markup is not supported***`	***nested markup is not supported***

Hyperlinks

External Links

Markup	Result
`http://www.python.org/`	http://www.python.org/
`External hyperlinks, like `Python <http://www.python.org/>`_`	External hyperlinks, like Python
`External hyperlinks, like Moin_.` `.. _Moin:  http://moinmo.in/`	External hyperlinks, like Moin.

Internal Links

Markup	Result
`http:Home` link to a page in this wiki	http:Home link to a page in this wiki
`` `Home <http:Home>`_ `` link to a page in this wiki	Home link to a page in this wiki
`Headings_` link to heading anchor on this page	*Headings* link to heading anchor on this page
`` `Internal Links`_ `` link to heading with embedded blanks	*Internal Links* link to heading with embedded blanks
`.. _myanchor:` create anchor, real anchor is above this table	create anchor, real anchor is above this table
`myanchor_` link to above anchor	*myanchor* link to above anchor

Notes:

- If this page was created by Sphinx, none of the above internal link examples work correctly.
- The ".. _myanchor:" directive must begin in column one.
- Section titles (or headings) automatically generate hyperlink targets (the title text is used as the hyperlink name).

Images

Images may be positioned by using the align parameter with a value of left, center, or right. There is no facility to embed an image within a paragraph. There must be a blank line before and after the image declaration. Images are not enclosed within a block level element so several images declared successively without any positioning will display in a horizontal row.

Markup:

```
Before text.

.. image:: png
   :height: 100
   :width: 200
   :scale: 50
   :alt: alternate text png
   :align: center

After text.
```

Result:

Before text.

After text.

Notes:

- The Sphinx parser does not have an image named "png" so the alternate text will be displayed.

Figures

Figures display graphics like images, but have the added feature of supporting captions and explanatory text. Figures are block elements, so figures declared successively will display in a column.

Markup:

```
Before text.

.. figure:: png
   :height: 100
   :width: 200
   :scale: 50
   :alt: alternate text png

   Moin Logo

   This logo replaced the "MoinMoin Man"
   logo long ago.

After text.
```

Result:

Before text.

Fig. 1: Moin Logo
This logo replaced the "MoinMoin Man" logo long ago.

After text.

Notes:

- The Sphinx parser does not have an image named "png" so the alternate text will be displayed.
- The Sphinx parser does not support figures so the caption and explanatory text will not display correctly.

Blockquotes and Indentations

To create a blockquote, indent all lines of a paragraph or paragraphs with an equal number of spaces. To add an attribution, begin the last indented paragraph with "– ".

Markup:

```
Text introducing a blockquote:

 If you chase two rabbits, you will lose them both.
```

Result:

Text introducing a blockquote:

If you chase two rabbits, you will lose them both.

Markup:

```
This is an ordinary paragraph, introducing a block quote.

  "It is my business to know things.  That is my trade."

  -- Sherlock Holmes
```

Result:

This is an ordinary paragraph, introducing a block quote.

> "It is my business to know things. That is my trade."
>
> —Sherlock Holmes

Lists

Unordered Lists

Markup:

```
- item 1
- item 2

  - item 2.1
  - item 2.2

    - item 2.2.1
    - item 2.2.2

- item 3
```

Result:

- item 1
- item 2
  - item 2.1
  - item 2.2
    - item 2.2.1
    - item 2.2.2
- item 3

Ordered Lists

Markup:

```
1. item 1
#. item 2

   (A) item 2.1
```

(continues on next page)

(continued from previous page)

```
   (#) item 2.2

       i) item 2.2.1
       #) item 2.2.2

#. item 3
```

Result:

1. item 1
2. item 2
   (A) item 2.1
   (B) item 2.2
       i) item 2.2.1
       ii) item 2.2.2
3. item 3

Notes:

- Ordered lists can be automatically enumerated using the `#` character as demonstrated above. Note that the first item of an ordered list auto-enumerated in this fashion must use explicit numbering notation (e.g. `1.`) in order to select the enumeration sequence type (e.g. Roman numerals, Arabic numerals, etc.), initial number (for lists which do not start at "1") and formatting type (e.g. `1.` or `(1)` or `1)`). More information on enumerated lists can be found in the reStructuredText documentation.
- One or more blank lines are required before and after reStructuredText lists.
- The Moin2 parser requires a blank line between items when changing indentation levels.
- Formatting types (A) and i) are rendered as A. and A. by Sphinx and as A. and i. by Moin2.

Definition Lists

Definition lists are formed by an unindented one line term followed by an indented definition.

Markup:

```
term 1
 Definition 1.

term 2 : classifier
 Definition 2.

term 3 : classifier one : classifier two
 Definition 3.
```

Result:

term 1 Definition 1.

term 2 [classifier] Definition 2.

term 3 [classifier one][classifier two] Definition 3.

Field Lists

Field lists are part of an extension syntax for directives usually intended for further processing.

Markup:

```
:Date: 2001-08-16
:Version: 1
:Authors: Joe Doe
```

Result:

Date 2001-08-16

Version 1

Authors Joe Doe

Option lists

Option lists are intended to document Unix or DOS command line options.

Markup:

```
-a      command definition
--a     another command definition
/S      dos command definition
```

Result:

-a	command definition
--a	another command definition
/S	dos command definition

Transitions

Transitions, or horizontal rules, separate other body elements. A transition should not begin or end a section or document, nor should two transitions be immediately adjacent. The syntax for a transition marker is a horizontal line of 4 or more repeated punctuation characters. The syntax is the same as section title underlines without title text. Transition markers require blank lines before and after.

Markup:

```
Text

----

Text
```

Result:

Text

Text

Backslash Escapes

Sometimes there is a need to use special characters as literal characters, but reST's syntax gets in the way. Use the backslash character as an escape.

Markup:

```
*hot*

333. is a float, 333 is an integer.

\*hot\*

333\. is a float, 333 is an integer.
```

Result:

hot

333. is a float, 333 is an integer.

hot

333. is a float, 333 is an integer.

Notes:

- The Moin2 reST parser changes the 333. to a 1. and inserts an error message into the document.
- The Sphinx reST parser begins an ordered list with 333. The visual effect is a dedented line.

Tables

Simple Tables

Easy markup for tables consisting of two rows. This syntax can have no more than two rows.

Markup:

```
======= ======= =======
 A       B       C
======= ======= =======
 1       2       3
======= ======= =======
```

Result:

A	B	C
1	2	3

Markup:

```
======= ======= =======
      foo          Bar
--------------- -------
 A       B       C
======= ======= =======
 1       2       3
======= ======= =======
```

Result:

foo		Bar
A	B	C
1	2	3

Grid Tables

Complex tables can have any number of rows or columns. They are made by |, +, − and =.

Markup:

```
+----------------+---------------+
| A              |               |
+----------------+ D             |
| B              |               |
+================+===============+
| C                              |
+--------------------------------+
```

Result:

A	D
B	
C	

One difference between the Sphinx and Moin reST parsers is demonstrated below. With the Spinx parser, grid table column widths can be expanded by adding spaces.

Markup:

```
+--------------+-------------------------------------------------------------------------------
↪-------------------------------------------------------------------------------------------
↪-----+
| minimal width | maximal width (will take the maximum screen space)                          ↪
↪     |
+---------------+-------------------------------------------------------------------------------
↪-------------------------------------------------------------------------------------------
↪-----+
```

Result:

minimal width	maximal width (will take the maximum screen space)

Notes:

- The Moin2 reST parser does not add the <colgroup><col width="9%"><col width="91%"> HTML markup added by the Sphinx parser (the width attribute generates an HTML validation error), nor does it use Javascript to adjust the width of tables.
- Under Moin2, tables and table cells will be of minimal width (unless there is CSS styling to set tables larger).

Admonitions

Admonitions are used to draw the reader's attention to an important paragraph. There are nine admonition types: attention, caution, danger, error, hint, important, note, tip, and warning.

The reST parser uses "error" admonitions to highlight some reST syntax errors.

Markup:

```
.. caution:: Be careful!
.. danger:: Watch out!
.. note:: Phone home.
```

Result:

Caution: Be careful!

Danger: Watch out!

Note: Phone home.

Comments

Comments are not shown on the page. Some parsers may create HTML comments (<!-- -->). The Sphinx parser suppresses comments in the HTML output. Within the Moin2 wiki, comments may be made visible/invisible by clicking the Comments link within item views.

Markup:

```
.. This is a comment
..
 _so: is this!
..
 [and] this!
..
 this:: too!
..
 |even| this:: !
```

Result:

Literal Blocks

Literal blocks are used to show text as-it-is. i.e no markup processing is done within a literal block. A minimum (1) indentation is required for the text block to be recognized as a literal block.

Markup:

```
Paragraph with a space before two colons ::

 Literal block

Paragraph with no space before two colons::

 Literal block
```

Result:

Paragraph with a space between preceding two colons

```
Literal block
```

Paragraph with no space between text and two colons:

```
Literal block
```

Line Blocks

Line blocks are useful for address blocks, verse (poetry, song lyrics), and unadorned lists, where the structure of lines is significant. Line blocks are groups of lines beginning with vertical bar ("|") prefixes. Each vertical bar prefix indicates a new line, so line breaks are preserved. Initial indents are also significant, resulting in a nested structure. Inline markup is supported. Continuation lines are wrapped portions of long lines; they begin with a space in place of the vertical bar. The left edge of a continuation line must be indented, but need not be aligned with the left edge of the text above it. A line block ends with a blank line.

Markup:

```
Take it away, Eric the Orchestra Leader!

   | A one, two, a one two three four
   |
   | Half a bee, philosophically,
   |     must, *ipso facto*, half not be.
   | But half the bee has got to be,
   |     *vis a vis* its entity.  D'you see?
   |
   | But can a bee be said to be
   |     or not to be an entire bee,
   |         when half the bee is not a bee,
   |             due to some ancient injury?
   |
   | Singing...
```

Result:

Take it away, Eric the Orchestra Leader!

A one, two, a one two three four

Half a bee, philosophically,
must, *ipso facto*, half not be.
But half the bee has got to be,
vis a vis its entity. D'you see?

But can a bee be said to be
or not to be an entire bee,
when half the bee is not a bee,
due to some ancient injury?

Singing...

Contents

4.2.4 Docbook XML Markup

This page shows the different features of our native DocBook support. A table of contents is automatically generated from section titles.

This content, describing the Docbook syntax, is written in reST. Instances where reST cannot duplicate the same rendering produced by Docbook are flagged with **reST NOTE**. The reST parser used by Moin and the parser used by Sphinx are different.

Lists

Itemized List

Markup::

```
<itemizedlist>
  <listitem>
    <para>Item 1
    </para>
  </listitem>
  <listitem>
    <para>Item 2
    </para>
```

(continues on next page)

(continued from previous page)

```
  </listitem>
  <listitem>
    <para> Item 3
    </para>
  </listitem>
</itemizedlist>
```

Results:

- Item 1
- Item 2
- Item 3

Ordered List

Markup::

```
<orderedlist numeration="lowerroman">
  <listitem>
    <para>One</para>
  </listitem>
  <listitem>
    <para>Two</para>
  </listitem>
  <listitem>
    <para>Three</para>
  </listitem>
  <listitem>
    <para>Four</para>
  </listitem>
</orderedlist>
```

Results:

i. One

ii. Two

iii. Three

iv. Four

reST NOTE: should show small roman numbers here

Simple text formatting

Markup::

```
<para>
<emphasis role="bold">This</emphasis> paragraph contains
<emphasis>some <emphasis>emphasized</emphasis> text</emphasis>
and a <superscript>super</superscript>script
and a <subscript>sub</subscript>script.
</para>
```

Results: **This** paragraph contains *some *emphasized* text* and a ^{super}script and a $_{sub}$script.

Quotes

Markup::

```
<para>This software is provided <quote>as is</quote>, without expressed
or implied warranty.
</para>
```

Results: This software is provided “as is”, without expressed or implied warranty.

Trademarks and Copyrights

Markup::

```
<para><trademark class='registered'>Nutshell Handbook</trademark> is a
registered trademark of O'Reilly Media, Inc.
</para><para>
<trademark class="copyright">2014 Joe Doe</trademark>
</para><para>
<trademark class="trade">Foo Bar</trademark> is an unregistered trademark.
</para><para>
<trademark class="service">Foo Bar</trademark> is an unregistered servicemark.
</para>
```

Results: Nutshell Handbook® is a registered trademark of O’Reilly Media, Inc.

© 2014 Joe Doe

Foo Bar™ is an unregistered trademark.

Foo Bar℠ is an unregistered servicemark.

Preformatted Data

Markup::

```
<screen><![CDATA[
<para>
My  preformatted      data.

Remove blanks from "] ] >" below:
</para>
] ] ></screen>
```

Results::

```
<para>
My  preformatted      data.

Remove blanks from "] ] >" below:
</para>
```

Links

Markup::

```
<link xlink:href="https://moinmo.in/">MoinMoin rocks</link>
```

Results:

MoinMoin rocks

Tables

Markup::

```
<table frame='all'><title>Sample Table</title>
<tgroup cols='5' align='left' colsep='1' rowsep='1'>
<colspec colname='c1'/>
<colspec colname='c2'/>
<colspec colname='c3'/>
<colspec colnum='5' colname='c5'/>
<thead>
<row>
  <entry namest="c1" nameend="c2" morecols='1' align="center">Horizontal Span</entry>
  <entry>a3</entry>
  <entry>a4</entry>
  <entry>a5</entry>
</row>
</thead>
<tfoot>
<row>
  <entry>f1</entry>
  <entry>f2</entry>
  <entry>f3</entry>
  <entry>f4</entry>
  <entry>f5</entry>
</row>
</tfoot>
<tbody>
<row>
  <entry>b1</entry>
  <entry>b2</entry>
  <entry>b3</entry>
  <entry>b4</entry>
  <entry morerows='1' valign='middle'><para>  <!-- Pernicous Mixed Content -->
  Vertical Span</para></entry>
</row>
<row>
  <entry>c1</entry>
  <entry namest="c2" nameend="c3" morecols='1' align='center' morerows='1' valign=
↪'bottom'>Span Both</entry>
  <entry>c4</entry>
</row>
<row>
  <entry>d1</entry>
  <entry>d4</entry>
  <entry>d5</entry>
</row>
</tbody>
</tgroup>
</table>
```

Results:

Horizontal Span		a3	a4	a5
b1	b2	b3	b4	Vertical Span
c1	Span Both		c4	
d1			d4	d5
f1	f2	f3	f4	f5

reST NOTE: the table does not show the correct result.

Images

An "inlinemediaobject" may be positioned within a paragraph and aligned to the text top, middle, or bottom through use of the align attribute.

Markup::

```
<para>
Here is an image
<inlinemediaobject>
  <imageobject>
    <imagedata format="png" align="middle" fileref="png"/>
  </imageobject><caption>My Logo</caption>
</inlinemediaobject>
embedded in a sentence.
</para>
```

Results:

Here is an image

embedded in a sentence.

Notes:

- The Sphinx parser does not have an image named "png" so the alternate text will be displayed.
- **reST NOTE**: There is no facility to embed an image within a paragraph.

Footnotes

All footnotes are placed at the bottom of the document in the order defined.

Markup::

```
<para>An annual percentage rate (<abbrev>APR</abbrev>) of 13.9%<footnote>
<para>The prime rate, as published in the Wall Street
Journal on the first business day of the month,
plus 7.0%.
</para>
</footnote>
will be charged on all balances carried forward.
</para>
```

Results:

An annual percentage rate (APR) of 13.9%[1] will be charged on all balances carried forward.

4.2.5 Mediawiki markup overview

Features currently not working with moin's mediawiki parser are marked with **MWTODO**.

Features currently not working with moin's rst parser are marked with **reSTTODO**.

Headings

Markup:

```
= Level 1 =
== Level 2 ==
=== Level 3 ===
==== Level 4 ====
===== Level 5 =====
====== Level 6 ======
```

Result:

Level 1

Intentionally not rendered as level 1 so it does not interfere with Sphinx's indexing

Level 2

Level 3

Level 4

Level 5

Level 6

Text formatting

These markups can be used within text to apply character style.

[1] The prime rate, as published in the Wall Street Journal on the first business day of the month, plus 7.0%.

Markup	Result
`'''Bold text'''`	**Bold text**
`''Italic text''`	*Italic text*
`'''''Bold and italic text'''''`	Bold and italic text
`<nowiki>no ''markup''</nowiki>`	`no ''markup''`
`<u>underline</u>`	underline
`<del>strikethrough</del>` or `<s>striketrough</s>`	strikethrough
`<code>Fixed width</code>` or `<tt>Fixed width</tt>`	`Fixed width`
`<pre>Preformatted text without '''markups'''</pre>`	`Preformatted text without '''markups'''`

reSTTODO table headers are not formatted as headers (see "Tables" section for corresponding MWTODO)

Hyperlinks

Internal links

reSTTODO These link targets are not interpreted. (The examples shown here result in empty links) **reSTTODO** Comments (lines starting with `..`) are printed

Markup	Result	Comment
`[[Item name]]`	*Item name*	Link to an item
`[[Item name|alternative text]]`	*alternative text*	Link with alternative text
`[[#anchor]]`	*#anchor*	Link to an anchor on this item
`[[#anchor|alternative text]]`	*alternative text*	Link to an anchor with alternative text
`[[Item name#anchor]]`	*Item name#anchor*	Link to an anchor on another item
`<div id="anchor">text</div>`	text	Definition of an anchor **MWTODO** (div tag is not interpreted)
`[[/subitem]]`	*/subitem*	Link to a subitem
`[[media:image.jpg]]`	*media:image.jpg*	Link to a file **MWTODO** (irrelevant for moin?)

External links

Markup	Result	Comment
`http://www.example.com`	http://www.example.com	External link **MWTODO** (not converted into a hyperlink)
`[http://www.example.com text]`	text	External link with alternative text
`[http://www.example.com]`	[1]	External link with number **MWTODO** (no numbering, normal link)
`[mailto:test@example.com mail]`	mail	Mailto link

Images

MWTODO Use of `[[File:...]]` causes this error: `AttributeError: 'unicode' object has no attribute 'keyword'`

Syntax

The syntax for inserting an image is as follows:

```
[[File:<filename>|<options>|<caption>]]
```

The *options* field can be empty or can contain one or more of the following options separated by pipes (`|`).

Format option: Controls how the image is formatted in the item.

one of `border` and/or `frameless`, `frame` or `thumb`

Resizing option: Controls the display size of the picture. The aspect ratio cannot be changed.

one of `<width>px`, `x<height>px`, `<width>x<height>px` or `upright`

Horizontal alignment option: Controls the horizontal alignment of an image.

one of `left`, `right`, `center` or `none`

Vertical alignment option: Controls the vertical alignment of a non-floating inline image.

one of `baseline`, `sub`, `super`, `top`, `text-top`, `middle` (default), `bottom` or `text-bottom`

Link option: The option `link=<target>` allows to change the target of the link represented by the picture. The image will not be clickable if `<target>` is left empty.

Please note that the link option cannot be used with one of the options `thumb` or `frame`.

Other options: The `alt=<alternative text>` option sets the alternative text (HTML attribute `alt=`) of the image.

The option `page=<number>` sets the number of the page of a .pdf or .djvu file to be rendered.

Examples

Markup	Description
`[[File:example.png]]`	Displays an image without further options.
`[[File:example.png|border]]`	Displays the image with a thin border.
`[[File:example.png|frame|text]]`	Displays the image in a frame (not inline) and shows *text* as caption.
`[[File:example.png|thumb|text]]`	Displays a thumbnail of the image (not inline) and shows *text* as caption.
`[[File:example.png|frameless]]`	Like `thumb` but inline and without border and frame

Paragraphs

Markup:

```
You can leave an empty line to start a new paragraph.

Single breaks are ignored.
To force a line break, use the <br /> HTML tag.
```

Result:

You can leave an empty line to start a new paragraph.

Single breaks are ignored. To force a line break, use the
HTML tag.

Horizontal rules

Markup:

```
A horizontal rule can be added by typing four dashes.

----

This text will be displayed below the rule.
```

Result:

A horizontal rule can be added by typing four dashes.

This text will be displayed below the rule.

reSTTODO Horizontal rule is not interpreted.

Preformatted text

Markup:

```
␣Each line that starts
␣with a space
␣is preformatted. It is ''possible''
␣to use inline '''markups'''.
```

Result:

Each line that starts
with a space
is preformatted. It is *possible*
to use inline **markups**.

MWTODO Preformatted text is not interpreted.

reSTTODO Line blocks (lines starting with `|`) are not interpreted.

Comments

Markup:

```
<!-- This is a comment -->
Comments are only visible in the modify window.
```

Result:

Comments are only visible in the modify window.

MWTODO This is not interpreted (i.e. comments are printed).

MWTODO A line starting with `##` is treated as comment, although it should be treated as part of an ordered list (see section "Ordered lists").

MWTODO It seems that `/*...*/` is treated as comment, whereas this is not intended in mediawiki syntax.

Symbol entities

A special character can be placed by using a symbol entity. The following table shows some examples for symbol entities:

Entity	Character
`&mdash;`	—
`&larr;`	←
`&rarr;`	→
`&lArr;`	
`&rArr;`	
`&copy;`	©

It is also possible to use numeric entities like `&#xnnnn;` where "nnnn" stands for a hexadecimal number.

Lists

Ordered lists

Ordered lists are formed of lines that start with number signs (#). The count of number signs at the beginning of a line determines the level.

Markup:

```
# First item
# Second item
## First item (second level)
## Second item (second level)
### First item (third level)
# Third item
```

Result:

1. First item
2. Second item
    1. First item (second level)
    2. Second item (second level)
        1. First item (third level)
3. Third item

Unordered lists

Markup:

```
* List item
* List item
** List item (second level)
*** List item (third level)
* List item
```

Result:

- List item
- List item
    - List item (second level)
    - List item (third level)
- List item

Definition lists

Markup:

```
;term
: definition
;object
: description 1
: description 2
```

Result:

term definition

object description 1

description 2

Mixed lists

It is possible to combine different types of lists.

Markup:

```
# first item
# second item
#* point one
#* point two
# third item
#; term
#: definition
#: continuation of the definition
# fourth item
```

Result:

1. first item
2. second item
   - point one
   - point two
3. third item

   term definition

   continuation of the definition
4. fourth item

Indentations

Definition lists can also be used to indent text.

Markup:

```
: single indent
:: double indent
:::: multiple indent
```

Result:

single indent

double indent multiple indent

Footnotes

Footnotes can be used for annotations and citations rolled out of the continuous text.

Markup:

```
This is a footnote <ref>This description will be placed at the item's bottom.</ref>
```

Result:

This is a footnote [1].

[1] This description will be placed at the item's bottom.

Tables

Syntax

Markup	Description
{|	**table start** (required)
|+	**table caption** (optional) **MWTODO** (not interpreted) only between table start and first row
|-	**table row** (optional) This is not necessary for the first row.
|	**table data** (required) Start each line that contains table data with | or separate data on the same line with ||
!	**table header** (optional) **MWTODO** (not formatted as header) Start each line that represents a table header with ! or separate different headers on the same line with !!.
|}	**table end** (required)

Basic tables

Note that the following tables do not have visible borders as this has to be done with XHTML attributes.

MWTODO Tables should be borderless by default, the `border` attribute is not interpreted.

Markup:

```
{|
|row 1, column 1
|row 1, column 2
|-
|row 2, column 1
|row 2, column 2
|}
```

Result:

row 1, column 1	row 1, column 2
row 2, column 1	row 2, column 2

Markup:

```
{|
!header 1
!header 2
|-
|A
|B
|-
|C
|D
|}
```

Alternative syntax:

```
{|
!header 1!!header 2
|-
|A||B
|-
|C||D
|}
```

Result:

header 1	header 2
A	B
C	D

It is possible to use other elements inside tables:

Markup:

```
{|
!header 1
!header 2
|-
|A line break<br />can be done with the XHTML tag.
|A pipe symbol has to be inserted like this: <nowiki>|</nowiki>
|-
|
* This
* is a bullet list
* in a table cell.
|[http://www.example.com Hyperlink]
|}
```

Result:

header 1	header 2
A line break can be done with the XHTML tag.	A pipe symbol has to be inserted like this: |
• This • is a bullet list • in a table cell	Hyperlink

MWTODO Lists cannot be used inside cells.

XHTML attributes

It is allowed to use XHTML attributes (border, align, style, colspan, rowspan, ...) inside tables.

Markup:

```
{|border="1"
|This table has a border width of 1.
|align="left" | This cell is left aligned.
|-
|colspan="2" | This cell has a colspan of 2.
|}
```

Result:

This table has a border width of 1.	This cell is left aligned.
This cell has a colspan of 2.	

MWTODO attributes `border` and `align` are not interpreted

reSTTODO colspan is not interpreted

4.2.6 Markdown Markup

This page introduces you to the most important elements of the Markdown syntax. For details on the Python implementation of Markdown see https://pythonhosted.org/Markdown/

In addition to being supported by moin2, the Markdown markup language is used by issue trackers such as those found in Bitbucket and Github. So what you learn here can be used there also.

Features currently not working with moin's Markdown parser are marked with **MDTODO**.

This page, describing the Markdown syntax, is written in reST. Instances where reST cannot duplicate the same rendering produced by Markdown are flagged with **reST NOTE**. The reST parser used by Moin and the parser used by Sphinx are different. As noted below there are several instances where one works and the other fails.

Table of Contents

The table of contents is a supported extension that is distributed with Python Markdown.

Markup:

```
[TOC]
```

Result:

Contents

Headings

Level 1 and 2 headings may be created by underlining with = and - characters, respectively.

Having equal numbers of characters in the heading and the underline looks best in raw text, but having fewer or more = or - characters also works.

Heading levels 3 through 6 must be defined by prefixing the heading with a variable number of # characters indicating the heading level. Heading levels 1 and 2 may be defined in the same manner. It is customary, but not required, to follow the # characters with a single space character. Another option is to append the appropriate number of # characters after the heading text.

Markup:

```
Level 1
=======

# Level 1

Level 2
-------

## Level 2

### Level 3

#### Level 4

##### Level 5

###### Level 6 ######
```

Result:

Level 3

Level 4

Level 5

Level 6

NOTE: Levels 1 and 2 are not shown above to avoid adding unwanted entries to the table of contents. See the top of this page for an approximate view of a level 1 heading and next section heading below for level 2.

Preformatted Code

To show a preformatted block of code, indent all the lines by 4 or more spaces.

Markup:

```
Begin preformatted code

   First line
   Second line
       Third line

End of preformatted code
```

Result:

Begin preformatted code

```
First line
Second line
    Third line
```

End of preformatted code

Simple text editing

Markup:

```
Paragraphs are separated
by a blank line.

To create a line break, end a line
with 2 spaces.

Use asterisk characters to create text attributes: *italic*, **bold**, ***bold␣
↪italics***.
Or, do the same with underscores: _Italics_, __bold__, ___bold italics___.
Use backticks to create `monospace`.
```

Result:

Paragraphs are separated by a blank line.

To create a line break, end a line
with 2 spaces.

Use asterisk characters to create text attributes: *italic*, **bold**, bold italics. Or, do the same with underscores: *Italics*, **bold**, bold italics. Use backticks to create `monospace`.

reST Note: The moin reST parser will indent the second paragraph above.

Linking

Markdown supports two style of links: inline and reference.

Inline Links

Inline links use the form:

```
[link text](url "optional title")
```

Markup	Result
[home page](Home)	home page
[home item](Home “my home page”)	home item
[a sub item](Home/subitem)	a sub item
[toc1](markdown#table-of-contents)	toc1
[toc2](#table-of-contents)	toc2
[moinmoin](https://moinmo.in “Go there”)	moinmoin
[![Image name](png)](Home “click me”)	png image

reST NOTE: Links with title attributes and images as links are not supported in reST. The internal links above are broken.

Reference Links

Reference links have two parts. Somewhere in the document the link label is defined using a unique id; this has no visible output. Then the reference link uses a form with square brackets rather than parens:

```
[id]: url "optional title"

[link text] [id]
```

Markup	Result
[apple]: https://www.apple.com/	
[MoinMoin]: https://moinmo.in/ "go!"	
[see apples][apple]	see apples
[go to MoinMoin][MoinMoin]	go to MoinMoin

reST NOTE: Links with title attributes are not supported in reST.

Lists

Unordered lists may use *, +, or - characters as bullets. The character used as a bullet does not effect the display. The display would be the same if * characters were used everywhere.

Markup:

```
* apples
* oranges
* pears
    - carrot
    - beet
        + man
        + woman
    - turnip
* cherries
```

Result:

- apples
- oranges
- pears
  - carrot
  - beet
    - man
    - woman
  - turnip
- cherries

reST NOTE: As shown above and below, the Sphinx rendering of ordered and unordered lists shows excessive spacing between levels.

Ordered lists use numbers and are incremented in regular order. Neither alpha characters nor roman numerals are supported. Although you may use numbers other than 1 with no adverse effect (as shown below), it is a best practice to always start a list with 1.

Markup:

```
1. apples
1. oranges
7. pears
    1. carrot
    1. beet
        1. man
        1. woman
    1. turnip
1. cherries
```

Result:

1. apples
2. oranges
3. pears
    1. carrot
    2. beet
        1. man
        2. woman
    3. turnip
4. cherries

Lists composed of long paragraphs are easier to read in raw text if the lines are manually wrapped with **optional** hanging indents. If multiple paragraphs are required, separate the paragraphs with blank lines and indent.

Markup:

```
*   Lorem ipsum dolor sit amet, consectetuer adipiscing elit.
    Aliquam hendrerit mi posuere lectus. Vestibulum enim wisi,
    viverra nec, fringilla in, laoreet vitae, risus.
*   Donec sit amet nisl. Aliquam semper ipsum sit amet velit.
    Suspendisse id sem consectetuer libero luctus adipiscing.
*   Lorem ipsum dolor sit amet, consectetuer adipiscing elit.
Aliquam hendrerit mi posuere lectus. Vestibulum enim wisi,
viverra nec, fringilla in, laoreet vitae, risus.
*   Lorem ipsum dolor sit amet, consectetuer adipiscing elit.
Aliquam hendrerit mi posuere lectus. Vestibulum enim wisi,
viverra nec, fringilla in, laoreet vitae, risus.
*   Donec sit amet nisl. Aliquam semper ipsum sit amet velit.
Suspendisse id sem consectetuer libero luctus adipiscing.
```

Result:

- Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Aliquam hendrerit mi posuere lectus. Vestibulum enim wisi, viverra nec, fringilla in, laoreet vitae, risus.

- Donec sit amet nisl. Aliquam semper ipsum sit amet velit. Suspendisse id sem consectetuer libero luctus adipiscing.
- Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Aliquam hendrerit mi posuere lectus. Vestibulum enim wisi, viverra nec, fringilla in, laoreet vitae, risus.
- Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Aliquam hendrerit mi posuere lectus. Vestibulum enim wisi, viverra nec, fringilla in, laoreet vitae, risus.
- Donec sit amet nisl. Aliquam semper ipsum sit amet velit. Suspendisse id sem consectetuer libero luctus adipiscing.

Horizontal Rules

To create horizontal rules, use 3 or more -, *, or _ on a line. Neither changing the character nor increasing the number of characters will change the width of the rule. Putting spaces between the characters also works.

Markup:

```
---

text

- - - - - -

more text

******

more text

_____
```

Result:

---

text

---

more text

---

more text

Backslash Escapes

Sometimes there is a need to use special characters as literal characters, but Markdown's syntax gets in the way. Use the backslash character as an escape.

Markup:

```
*hot*

333. is a float, 333 is an integer.

\*hot\*

333\. is a float, 333 is an integer.
```

Result:

hot

333. is a float, 333 is an integer.

hot

333. is a float, 333 is an integer.

reST NOTE: The Moin reST parser flags the use of 333 as a bullet number.

Nested Blockquotes

Advanced blockquotes with nesting are created by starting a line with a > character.

Markup:

```
> A standard blockquote is indented
> > A nested blockquote is indented more
> > > You can nest to any depth.
```

Result:

> **A standard blockquote is indented**
>
> > **A nested blockquote is indented more** You can nest to any depth.

Images

Images are similar to links with both an inline and a reference style, but they start with an exclamation point. Within Markdown, there is no syntax to change the default sizes or positions of transclusions:

Markup:

```
To transclude image from local wiki:
![Alt text 1](png "Optional title")

Reference-style, where "logo" is a name defined anywhere within this item:
![Alt text 2][logo]

Image references are defined using syntax identical to link references and
do not appear in the rendered HTML:
[logo]: png  "Optional title attribute"

To transclude image from remote site:
![remote image](http://static.moinmo.in/logos/moinmoin.png)
```

Result:

To transclude image from local wiki:

Reference-style, where "logo" is a name defined anywhere within this item:

Image references are defined using syntax identical to link references and do not appear in the rendered HTML:

To transclude image from remote site:

reST NOTE: The Moin reST parser renders all three images above. The Sphinx parser renders only the external png image from http://static.moinmo.in/logos/moinmoin.png. reST syntax does not allow the rendering of inline images, nor the use of a title attribute. The logos above are floated right, in Markdown the logos would appear as inline images.

Inline HTML

Note: Use of the style attribute within HTML tags is dependent upon configuration settings. See configuration docs for information on *allow_style_attributes*.

You may embed a small subset of HTML tags directly into your markdown documents.

```
<a>              - hyperlink.
<b>              - bold, use as last resort <h1>-<h3>, <em>, and <strong> are␣
↪preferred.
<blockquote>     - specifies a section that is quoted from another source.
<code>           - defines a piece of computer code.
<del>            - delete, used to indicate modifications.
<dd>             - describes the item in a <dl> description list.
<dl>             - description list.
<dt>             - title of an item in a <dl> description list.
<em>             - emphasized.
<h1>, <h2>, <h3> - headings.
<i>              - italic.
<img>            - specifies an image tag.
<kbd>            - shows keyboard input.
<li>             - list item in an ordered list <ol> or an unordered list <ul>.
<ol>             - ordered list.
<p>              - paragraph.
<pre>            - pre-element displayed in a fixed width font and unchanged line␣
↪breaks.
<s>              - strikethrough.
<sup>            - superscript text appears 1/2 character above the baseline used for␣
↪footnotes and other formatting.
<sub>            - subscript appears 1/2 character below the baseline.
<strong>         - defines important text.
<strike>         - strikethrough is deprecated, use <del> instead.
<ul>             - unordered list.
<br>             - line break.
<hr>             - defines a thematic change in the content, usually via a horizontal␣
↪line.
```

Markup:

```
E = MC<sup>2</sup>

This word is <b>bold</b>.

This word is <em>italic</em>.

This word is <strong>bold</strong>.

This word is <strong style="color:red;background-color:yellow">bold</strong>;
colors depend upon configuration settings.
```

Result:

reST NOTE: The moin reST parser will flag the above as an error because it does not support the *raw* directive.

Extensions

In addition to the TOC extension shown near the top of this page, the following features are installed as part of the "extras" extension.

Tables

All tables must have one heading row. By default table headings are centered and table body cells are aligned left. Use a ":" character on the left, right or both sides of the heading-body separator to change the alignment. Changing the alignment changes both the heading and table body cells.

As shown in the second table below, use of outside borders and neat alignment of the cells do not effect the display. Markup within the table cells is supported.

Markup:

```
|Tables            |Are            |Very  |Cool    |
|------------------|:-------------:|-----:|:-------|
|col 2 is          |centered       |$12   |Gloves  |
|col 3 is          |right-aligned  |$1600 |Necklace|
|col 4 is          |left-aligned   |$100  |Hat     |

`Tables`            |*Are*            |Very  |Cool
------------|:-------------:|-----:|:-------
`col 2 is`|*centered*|$12|Gloves
`col 3 is`|*right-aligned*|$1600|Necklace
`col 4 is`|*left-aligned*|$100|Hat
```

Result:

Tables	Are	Very	Cool
col 2 is	centered	$12	Gloves
col 3 is	right-aligned	$1600	Necklace
col 4 is	left-aligned	$100	Hat

Tables	*Are*	Very	Cool
col 2 is	*centered*	$12	Gloves
col 3 is	*right-aligned*	$1600	Necklace
col 4 is	*left-aligned*	$100	Hat

reST NOTE: reST does not support cell alignment.

Syntax Highlighting of Preformatted Code

A second way to create a block of preformatted code without indenting every line is to wrap the block in triple backticks.

To highlight code syntax, wrap the code in triple backtick characters and specify the language on the first line. Many languages are supported.

Markup:

```
``` javascript
var s = "JavaScript syntax highlighting";
alert(s);
```

~~~ {python}
def hello():
   print "Hello World!"
~~~
```

Result:

```
var s = "JavaScript syntax highlighting";
alert(s);

def hello():
   print "Hello World!"
```

reST NOTE: reST supports some generic highlighting of indented blocks. The Moin Markdown highlighting is more colorful and varies per language.

Fenced Code

Another way to display a block of preformatted code is to "fence" the code with lines starting with three ~ characters.

Markup:

```
~~~
ddd
eee
fff
~~~
```

Result:

```
ddd
eee
fff
```

Smart Strong

The smart strong extension prevents words with embedded double underscores from being converted. e.g. *double__underscore__words* is wanted, not *double‘**underscore**‘words*.

Markup:

```
Text with double__underscore__words.

__Strong__ still works.

__this__works__too__.
```

Result:

Text with double__underscore__words.

Strong still works.

this__works__too.

Attribute Lists

Markup:

```
A class of LawnGreen  (that will create a greenish background per a CSS rule) is
added to this paragraph.
{: class="LawnGreen "}

A `{: #para3 }` id was added to the 3rd paragraph on this page,
so [click to see 3rd paragraph](#para3).
```

Result:

A *{: #para3 }* id was added to the 3rd paragraph on this page, so click to see 3rd paragraph.

reST NOTE: The moin reST parser will flag the first example above as an error because it does not support the *raw* directive.

Definition Lists

Markup:

```
Apple
:   Pomaceous fruit of plants of the genus Malus in
    the family Rosaceae.
:   An american computer company.

Orange
:   The fruit of an evergreen tree of the genus Citrus.
```

Result:

Apple Pomaceous fruit of plants of the genus Malus in the family Rosaceae.

An american computer company.

Orange The fruit of an evergreen tree of the genus Citrus.

Footnotes

The syntax for footnotes in Markdown is rather unique.[^unique] Place any unique label after the characters "[^" and close the label with a "]". The footnote text may be placed after the reference on a new line using the label, followed by a ":", followed by the footnote text. All footnotes are placed at the bottom of the document under a horizontal rule in the order defined.

[^unique]: Markdown footnotes are unique.

Markup:

```
Footnotes[^1] have a label[^label] and a definition[^!DEF].

[^1]: This is a footnote
[^label]: A footnote on "label"
[^!DEF]: The footnote for definition
```

Result:

Footnotes[1] have a label[2] and a definition[3].

In Moin2, you specify the item's markup language when you create a new item. Currently Moin2 supports MoinWiki, WikiCreole, reStructuredText, Docbook, MediaWiki and Markdown markups.

Moin2 currently provides output converters for MoinWiki, Markdown, reST, HTML, and Docbook. When viewing any markup item, the item may be converted to a different markup language by clicking the Convert link.

4.3 Templates and Meta Data

Two features that are related to the creation, editing and saving of an item are templates and meta data.

4.3.1 Templates

Templates make it easier for users to add new items that repeat a heading structure or phrases that are similar to many other items. Instead of starting from scratch or using a copy, paste, delete, type new data; templates can be created that have the common text and structure already completed. A template page must have a tag of "template".

When creating a new item, users can save time and possibly eliminate errors by choosing a template from a list of applicable templates. Only templates having the same content type and namespace as the new item will be shown as choices.

For templates with the MoinWiki markup, Predefined Variables can be used to insert date, time, user name, item name, and others.

4.3.2 Meta Data

When an item is edited (including non-text items like images, etc.), most themes provide a means of updating certain meta data associated with the item. The meta data fields that may be updated by all editors include Summary, Tags, and Names.

Users with admin authority on the item may update the item's ACLs. The format of ACL rules is discussed within the configuration section under authorization.

[1] This is a footnote
[2] A footnote on "label"
[3] The footnote for definition

Most themes will display the Summary field above the item's content. The use of this field is optional. When used, it may contain a one-line summary of the pages content, a TODO list of additional content that should be added or verified, or other special instructions to future readers and editors.

For fields that may have multiple entries like the Tags and Names fields, use commas to separate the entries. Leading and trailing spaces are stripped, embedded spaces will become part of the tag or name.

Tags provide an alternate means of indexing articles. While tags are frequently used to group items based on the item's subject matter, they can also be used to group items in ways unrelated to the subject matter such as marking items that need additional content, editing, review, etc. Most themes provide a link to a Tags view within the navigation panel.

While most items will have a single name, item editors may add or delete multiple names. Editors may find multiple names useful when renaming or merging items. Item names cannot span multiple namespaces. Most themes will show all item names within the Page Trail panel. Some reports, such as History, will show all item names in a single row. Other reports which are sorted by name, such as Index and Tags, will show each name in a separate row.

4.4 Searching and Finding

4.4.1 Entering search queries

To start a search, enter a query into the short query input field and type enter or click the search icon. By default, the names, summary, tags, content, namengram, summaryngram, and contentngram fields are searched.

The search results view provides a form for refining the search through ajax updates. A transaction is started each time a character is added or removed in the search field. If keying is rapid, it is possible that results will processed out of order. The *Whoosh query* shows the last term processed.

Clicking the Search Options link displays alternatives for modifying the search. Ajax updates will be made whenever a radio button or checkbox is changed.

Below the search form is the query processed by Whoosh, and Whoosh generated suggestions for additional searches by input, item name, and item content.

Finally, the search hits are presented. By default these are ordered by the whoosh scoring number. Each hit will contain the item name and some meta data. If available, the item summary and partial item content with the search term highlighted will be shown.

4.4.2 Simple search queries

Just enter one or more words into the query input field and hit `Enter`.

If your query consists of multiple words, it will only find documents containing ALL those words. You can use AND, OR, NOT to refine your search. "AND" is the default.

Examples

Search for "wiki":

```
wiki
```

Search for documents containing "wiki" AND "moin":

```
wiki moin
```

Explicit alternative (does the same as above):

```
wiki AND moin
```

Search for documents containing “wiki” OR “moin”:

```
wiki OR moin
```

Search for documents containing “wiki” and NOT “bad”:

```
wiki NOT bad
```

Explicit alternative (does the same as above):

```
wiki AND NOT bad
```

Group terms using ():

```
wiki AND NOT (bad OR worse)
```

4.4.3 Using wildcards

If you want to enter word fragments or if you are not sure about spelling or word form, you can use wildcards for the parts you do not know:

Wildcard	Matches
?	one arbitrary character
*	any count of arbitrary characters

Examples

Search for something like wiki, wika, wikb, . . . :

```
wik?
```

Search for something like wiki, willi, wi, . . . :

```
w*i
```

You can also use it for poor man’s language independent word stemming.

Matches on clean, cleaner, cleanest, cleaning, . . . :

```
clean*
```

4.4.4 Using regular expressions

Regular expressions enable even more flexibility for specifying search terms.

See http://en.wikipedia.org/wiki/Regular_expression for basics about regexes.

See http://docs.python.org/library/re.html about python’s regex implementation, which we use for MoinMoin.

You need to use this syntax when entering regexes: r”yourregex”

Examples

Search for hello or hallo:

```
r"h[ae]llo"
```

Search for words starting with foo:

```
r"^foo"
r"\Afoo"
```

Search for something like wiki, wika, wikb, . . . :

```
r"wik."
```

Search for something like wiki, willi, wi, . . . :

```
r"w.*i"
```

4.4.5 Searching in specific fields

If not specified otherwise, moin will search in `names`, `tags`, `summary`, `comment` and `content` fields. Three fields with n-gram support are also searched by default: `namengram`, `summaryngram` and `contentngram`.

N-gram indexing is a powerful method for getting fast, "search as you type" functionality. A tokinizer splits words within ngram content fields into strings of 3 to 6 characters. These small strings may be matched against search terms that are tokinized into strings of 3 to 6 characters.

To specify the field to search in, just use the *fieldname:searchterm* syntax. If embedded spaces are desired then do: *fieldname:"search term"*. Separate multiple terms with a space: *content:foo tags:Foo* is the same as *content:foo AND tags:Foo*.

The following table includes fields that may be useful for searching.

Field name	Field value
`acl` **	access control list (see below)
`address`	submitter IP address, e.g. 127.0.0.1
`comment`	editor comment on save, rename, etc.
`content`	document contents, e.g. This is some example content.
`contentngram` **	document contents, tokenized by 3 to 6 characters.
`contenttype`	document type: text, image, audio, moinwiki, jpg, . . .
`itemlinks` **	link targets of the document, e.g. OtherItem
`itemtransclusions` **	transclusion targets of the document, e.g. OtherItem
`language`	(main) language of the document contents, e.g. en
`mtime`	document modification (submission) date, 2011-08-07
`namengram` **	document names, tokenized by 3 to 6 characters.
`names`	document names, e.g. Home, MyWikiPage
`namespace`	namespace:"" for default or namespace:users
`name_exact`	same as `name`, but is not tokenized
`name_old`	name_old:* for all renamed items
`summary`	summary text, if provided by author
`summaryngram` **	summary text, tokenized by 3 to 6 characters.
`tags`	tags of the document, e.g. important, hard, todo
`username`	submitter user name, e.g. JoeDoe
`wikiname`	wiki name, e.g. ITWiki, EngineeringWiki, SalesWiki

** These fields exist only in the current revisions index, see Notes below.

Examples

Search in metadata fields:

```
contenttype:text
contenttype:image/jpeg
tags:todo
mtime:2022-01-08  # use ISO 8601 dates, not time; `mtime:2022-01 works
address:127.0.0.1
username:JoeDoe
```

Search items with an item ACL that explicitly gives Joe read rights:

```
acl:Joe:+read
```

Limiting search to a specific wiki, for example in a wiki farm's shared index:

```
wikiname:SomeWiki  # requires correct caps
```

4.4.6 Notes

There are two indexes. The smaller index is used by default. It only indexes the current revision of each item. The larger index is used when the *All* radio button under the Search Options link is selected. The larger indexes all revisions of all items including revisions of deleted items. As noted in the table above the larger index omits several fields to save space.

By default, all namespaces and all wikinames are searched, including the userprofiles index. Because the userprofiles index is normally read restricted, hits will be blocked and included as *n items are not shown because read permission was denied* at the bottom of the page.

Items with transcluded content do not contain the transcluded content within the item's index. An item containing "foo" within its content and trancluding an item with "bar" within its content cannot be matched by searching for "foo AND bar". Both items will be matched by searching for "foo OR bar".

Moin only uses an indexed search. Keep in mind that this has some special properties:

- By using an index, the search is fast
- Because it is only using an index, it can only find what was put there
- If you use wildcards or regexes, it will still use the index, but in a different, slower way

For example:

- create an item with "FooBar" in the name, content, summary, tag, and comment fields
- search for "ooba" - the namengram, summaryngram, and contentngram will match
- search for "FooBar": names, namengram, tags, summary, summaryngram, content, contentngram, and comment will match
- search for "foobar": names, namengram, summary, summaryngram, content, contentngram, and comment will match

4.4.7 More information

See the Whoosh query language docs.

4.5 File Upload

File upload functionality is accessed through the +modify item view or the Index view.

To upload a file on the +modify item view, click the browsers Browse/Choose button. Use the browser's file dialog to select an item, then click the OK button. The file will be uploaded and saved with the previously chosen content type; file name suffixes are ignored.

To upload a file or files on the global index navigation view, start by clicking the *New Item* link to bring the *Create new item* dialog into view.

From the Index view, there are two methods of uploading files, single file or multiple files. Uploaded files will be logically placed within the current index, sub-index or namespace. Multiple file uploads have several restrictions:

- the files will be uploaded and saved using the current name
- existing items with the same name will be overwritten
- the file names should have a valid suffix that defines the file type
- files without a known suffix will be stored as is and available for download

4.5.1 Single File Upload

Enter the new item name into the input area and click the *Create* button. Select the content type to proceed to the +modify view. Use the browsers file dialog to select an item, then click the OK button. The file will be uploaded and saved with the chosen content type; file name suffixes are ignored.

4.5.2 Multiple File Upload

Click the browsers Browse/Choose button and select one or more files from the browser's file dialog or use the drag & drop method to copy files.

The file uploads will start immediately. Upload status will be displayed by overall and individual progress bars. The names of the files successfully uploaded will be prepended to list of files in the index.

4.6 Namespaces

MoinMoin supports the use of multiple namespaces where each namespace may have a unique backend or media type. For example, the default namespace could use the OS filesystem for item storage and another namespace could use an SQL database.

- an item in one namespace can readily include or transclude content from an item residing in another namespace.
- it is not possible for an item to have an alias name referencing a different namespace.
- it is not possible to rename an item into a different namespace.
- it is not possible to use a namespace name as an item name in a different namespace.

See the namespace section within MoinMoin configuration for information on how to configure namespaces.

4.6.1 URL layout

`http://server/[NAMESPACE/][[@FIELD/]VALUE][/+VIEW]`

Above defines the URL layout, where uppercase letters are variable parts defined below and [] denotes optional. It basically means search for the item field `FIELD` value `VALUE` in the namespace `NAMESPACE` and apply the view `VIEW` on it.

NAMESPACE Defines the namespace for looking up the item. NAMESPACE value `all` is the "namespace doesn't matter" identifier. It is used to access global views like global history, global tags etc.

FIELD Whoosh schema field where to lookup the VALUE (default: `name_exact`, lookup by name). FIELD can be a unique identifier like (`itemid, revid, name_exact`) or can be non-unique like (`tags`).

VALUE Value to search in the FIELD (default: the default root within that namespace). If the FIELD is non-unique, we show a list of items that have `FIELD:VALUE`.

VIEW used to select the intended view method (default: `show`).

Examples: The following examples show how a url can look like, `ns1, ns1/ns2` are namespaces.

- `http://localhost:8080/Home`
- `http://localhost:8080/ns1/@tags/sometag`
- `http://localhost:8080/ns1/ns2`
- `http://localhost:8080/ns1/SomePage`
- `http://localhost:8080/+modify/ns1/ns2/SomePage`
- `http://localhost:8080/+delete/ns1/@itemid/37b73d2a6c154bb4ab993d0fb463219c`
- `http://localhost:8080/ns1/@itemid/37b73d2a6c154bb4ab993d0fb463219c`

4.7 User Subscriptions

Users can subscribe to moin items in order to receive notifications about item changes. Item changes include:

- creation of a new item
- modification of an existing item
- renaming of an item
- reverting an item's revision
- copying of an item
- deletion of an item
- destruction of a revision
- destruction of all item's revisions

Make sure that Moin is able to send E-Mails, see *Mail configuration*.

4.7.1 Types of subscriptions

There are 5 types of subscriptions:

- by itemid (*itemid:<itemid value>*)

  This is the most common subscription to a single item. The user will be notified even after the item is renamed, because itemid doesn't change. If you click on *Subscribe* on item's page, then you will be subscribed using this type.

- by item name (*name:<namespace>:<name value>*),

  The user will be notified, if the name matches any of the item names and also its namespace. Keep in mind that an item can be renamed and notifications for this item would stop working if the new name doesn't match any more.

- by tag name (*tags:<namespace>:<tag value>*)

  The user will be notified, if the tag name matches any of the item tags and its namespace.

- by a prefix name (*nameprefix:<namespace>:<name prefix>*)

  Used for subscription to a set of items. The user will be notified, if at least one of the item names starts with the given prefix and matches item's namespace. For example if you want to receive notifications about all the items from the default namespace whose name starts with *foo*, you can use *nameprefix::foo*.

- by a regular expression (*namere:<namespace>:<name regexp>*)

  Used for subscription to a set of items. The user will be notified, if the regexp matches any of the item names and also its namespace. For example, if you want to receive notifications about all the items on wiki from the default namespace, then you can use *namere::.**

4.7.2 Editing subscriptions

The itemid subscription is the most common one and will be used if you click on *Subscribe* on item's page. Respectively the *Unsubscribe* will remove the itemid subscription.

If you were subscribed to an item by some other way rather than itemid subscription, then on *Unsubscribe* you will be told that it is impossible to remove the subscription and you need to edit it manually in the User Settings.

All the subscriptions can be added/edited/removed in the User Settings, Subscriptions tab. Each subscription is listed on a single line and is case-sensitive. Empty lines are ignored.

For itemid subscriptions, we additionally show the current first item name in parentheses (this is purely for your information, the name is not stored or used in any way).

CHAPTER 5

Administrating MoinMoin

5.1 Requirements

MoinMoin requires Python 3.8+. A CPython distribution is recommended because it will likely be the fastest and most stable. Most developers use a CPython distribution for testing. Typical linux distributions will either have Python 3.8+ installed by default or will have a package manager that will install Python 3.8+ as a secondary Python. Windows users may download CPython distributions from http://www.python.org/ or http://www.activestate.com/.

An alternative implementation of Python, PyPy, is available from http://pypy.org/.

5.1.1 Servers

For moin2, you can use any server compatible with WSGI:

- the builtin server (used by the "moin moin" command) is recommended for desktop wikis, testing, debugging, development, adhoc-wikis, etc.
- apache with mod_wsgi is recommended for bigger/busier wikis.
- other WSGI-compatible servers or middlewares are usable
- For cgi, fastcgi, scgi, ajp, etc., you can use the "flup" middleware: http://trac.saddi.com/flup
- IIS with ISAPI-WSGI gateway is also compatible: http://code.google.com/p/isapi-wsgi/

Caution: When using the built-in server for public wikis (not recommended), use "moin moin -D -R" to turn off the werkzeug debugger and auto reloader. See the Werkzeug docs for more information.

5.1.2 Dependencies

Dependent packages will be automatically downloaded and installed during the moin2 installation process. For a list of dependencies, see setup.py.

5.1.3 Clients

On the client side, you need:

- a web browser that supports W3C standards HTML 5, CSS 2.1, and JavaScript:
  - any current version of Firefox, Chrome, Opera, Safari, Maxthon, Internet Explorer (IE9 or newer).
  - use of older Internet Explorer versions is not recommended and not supported.
- a Java browser plugin is required only if you want to use the TWikiDraw or AnyWikiDraw drawing applets.

5.2 Installation

5.2.1 Installing the code

There are a lot of ways to do this and as this is not moin specific, we won't go into details:

- Use your operating system's / distribution's package manager to install the moin2 package. This is the recommended method as it will install moin2 and all other software it requires. Also your OS / dist might have a mechanism for updating the installed software with security fixes or to future releases.

  E.g. on Debian/Ubuntu Linux: `apt install moin2`
- Install from PyPI: `pip install moin2`
  - Optionally, create a virtual env first for better separation or
  - use `pip install --user moin2` to install into your home directory.
  - pip will automatically install other python packages moin2 requires, but you maybe have to install required non-python packages yourself.
  - You will have to care for updates / installing security fixes yourself.

After this, you should have a `moin` command available, try it:

```
moin --help
```

5.2.2 Creating a wiki instance

You'll need one instance directory per wiki site you want to run using moin - this is where wiki data, indexes and configuration for that site are stored.

Let's create a new instance:

```
moin create-instance INSTANCE-DIRECTORY
```

Change into the new instance directory:

```
cd INSTANCE-DIRECTORY
```

You'll find a `wikiconfig.py` there to edit. Adapt it as you like, you'll find some comments in there. Review and change the settings for:

- sitename
- interwikiname

- acls
- SECRET_KEY

After configuring, you can create an empty wiki by initializing the storage and the index:

```
moin index-create -s -i
```

If you don't want to start with an empty wiki, but rather play with some sample content we provide, load it into your wiki and rebuild the indexes:

```
moin load-sample
moin index-build

Or, if you have a moin 1.9.x wiki, convert it to moin 2:

::

 moin import19 -d <path to 1.9 wiki/data> -s -i
```

If you want to load English help for editors (replace en with your wiki's preferred language):

```
moin load-help -n en
moin load-help -n common
```

5.2.3 Run your wiki instance

Now try your new wiki using the builtin python-based web server:

```
moin moin  # visit the URL it shows in the log output
```

For production, please use a real web server like apache or nginx.

For more information on various wiki admin activities, see *Moin Command Line Interface*.

5.3 Installation (for developers)

5.3.1 Clone the git repository

If you like to work on the moin2 code, clone the master project repository or see the option below:

```
cd <to the parent of your moin repo>
git clone https://github.com/moinwiki/moin
cd moin
```

If you use github, you can also first fork the project repo to your own user's github repositories and then clone your forked repo to your local development machine. You can easily publish your own changes and do pull requests that way. If you do fork the project, then an alternative to the above command is to clone your fork and add a remote url to the master:

```
git clone https://github.com/<your name>/moin
cd moin
git remote add moinwiki https://github.com/moinwiki/moin
```

5.3.2 Installing

Before you can run moin, you need to install it.

Using your standard user account, run the following command from the project root directory. Replace <python> in the command below with the path to a python 3.8+ executable. This is usually just "python", but may be "python3", "python3.8", "/opt/pypy/bin/pypy" or even <some-other-path-to-python>:

```
<python> quickinstall.py

OR

<python> quickinstall.py <path-to-venv>
```

The above will download all dependent packages to the PIP cache, install the packages in a virtual environment, and compile the translations (*.*po* files) to binary *.*mo* files. This process may take several minutes.

The default virtual environment directory name is:

- ../<PROJECT>-venv-<PYTHON>/

where <PROJECT> is the name of the project root directory, and <PYTHON> is the name of your python interpreter. As noted above, the default name may be overridden.

Check the output of quickinstall.py to determine whether there were fatal errors. The output messages will normally state that stdout and stderr messages were written to a file, a few key success/failure messages will be extracted and written to the terminal window, and finally a message to type "m" to display a menu.

If there are failure messages, see the troubleshooting section below.

Activate the virtual environment:

```
activate #windows
. activate  # unix
```

Typing "./m" (or "m" on Windows) will display a menu similar to:

```
usage: "./m <target>" where <target> is:

quickinstall    update virtual environment with required packages
extras          install packages required for docs and moin development
docs            create moin html documentation (requires extras)
interwiki       refresh contrib/interwiki/intermap.txt (version control)
log <target>    view detailed log generated by <target>, omit to see list

new-wiki        create empty wiki
sample          create wiki and load sample data
restore *       create wiki and restore wiki/backup.moin *option, specify file
import19 <dir>  import a moin 1.9 wiki/data instance from <dir>

run *           run built-in wiki server *options (--port 8081)
backup *        roll 3 prior backups and create new backup *option, specify file
dump-html *     create a static HTML image of wiki *options, see docs
index           delete and rebuild indexes

css             run lessc to update basic theme CSS files
tests *         run tests, log output (-v -k my_test)
coding-std      correct scripts that taint the repository with trailing spaces..

del-all         same as running the 4 del-* commands below
```

(continues on next page)

(continued from previous page)

```
del-orig        delete all files matching *.orig
del-pyc         delete all files matching *.pyc
del-rej         delete all files matching *.rej
del-wiki        create a backup, then delete all wiki data
```

While most of the above menu choices may be executed now, new users should do the following to create a wiki instance and load it with sample data.:

```
m sample    # in Windows
./m sample # in Unix

If you want to load English help for editors (replace en with your wiki's preferred␣
↪language):
```

```
moin load-help -n en
moin load-help -n common
```

Next, run the built-in wiki server:

```
m run       # in Windows
./m run     # in Unix
```

As the server starts, a few log messages will be output to the terminal window. Point your browser to http://127.0.0.1:8080, the sample Home page will appear and more log messages will be output to the terminal window. Do a quick test by accessing some of the demo items and do a modify and save. If all goes well, your installation is complete. The built-in wiki server may be stopped by typing ctrl-C in the terminal window.

5.3.3 Next Steps

If you plan on contributing to the moin2 project, there are more instructions waiting for you under the Development topic.

If you plan on using this wiki as a production wiki, then before you begin adding or importing data and registering users review the configuration options. See the sections on configuration for details. Be sure to edit *wikiconfig.py* and change the settings for:

```
* sitename
* interwikiname
* acls
* SECRET_KEY
```

If you plan on just using moin2 as a desktop wiki (and maybe help by reporting bugs), then some logical menu choices are:

```
* `./m extras` - to install packages required for docs and moin development
* `./m docs` - to create docs, see User tab, Documentation (local)
* `./m del-wiki` - get rid of the sample data
* `./m new-wiki` or `m import19 ...` - no data or moin 1.9 data
* `./m backup` - backup wiki data as needed or as scheduled
```

If you installed moin2 by cloning the repository, then you will likely want to keep your master branch uptodate:

```
git checkout master
git pull # if you cloned the moinwiki master repo OR
git pull moinwiki master # if you cloned your fork and added a remote
```

After pulling updates, it is best to also rerun the quickinstall process to install any changes or new releases to the dependant packages:

```
m quickinstall  # in Windows
./m quickinstall # in Unix
```

5.3.4 Troubleshooting

PyPi down

Now and then, PyPi might be down or unreachable.

There are mirrors b.pypi.python.org, c.pypi.python.org, d.pypi.python.org you can use in such cases. You just need to tell pip to do so:

```
# put this into ~/.pip/pip.conf
[global]
index-url = http://c.pypi.python.org/simple
```

Bad Network Connection

If you have a poor or limited network connection, you may run into trouble with the commands issued by the quickinstall.py script. You may see tracebacks from pip, timeout errors, etc. within the output of the quickinstall script.

If this is the case, you may try rerunning the "python quickinstall.py" script multiple times. With each subsequent run, packages that are all ready cached (view the contents of pip-download-cache) will not be downloaded again. Hopefully, any temporary download errors will cease with multiple tries.

Other Issues

If you encounter some other issue not described above, try researching the unresolved issues in our issue tracker.

If you find a similar issue, please add a note saying you also have the problem and add any new information that may assist in the problem resolution.

If you cannot find a similar issue please create a new issue. Or, if you are not sure what to do, join us on IRC at #moin-dev and describe the problem you have encountered.

5.4 Server Options

5.4.1 Built-in Web Server (easy)

Moin comes with a simple built-in web server powered by Werkzeug, which is suitable for development, debugging, and personal and small group wikis.

It is *not* made for serving bigger loads, but it is easy to use.

Please note that by default the built-in server uses port 8080. As this is above port 1024, root (Administrator) privileges are not required and we strongly recommend that you use a normal, unprivileged user account instead. If you are running a desktop wiki or doing moin development, then use your normal login user.

Running the built-in server

Run the moin built-in server as follows:

```
# easiest for debugging (single-process, single-threaded server):
moin moin

# or, if you need another configuration file, ip address, or port:
moin --config /path/to/wikiconfig.py moin --host 1.2.3.4 --port 7777
```

While the moin server is starting up, you will see some log output, for example:

```
2016-01-11 13:30:05,394 INFO werkzeug:87  * Running on http://127.0.0.1:8080/ (Press␣
↪CTRL+C to quit)
```

Now point your browser at that URL - your moin wiki is running!

Stopping the built-in server

To stop the wiki server, either use *Ctrl-C* or close the window.

Debugging with the built-in server

Werkzeug has a debugger that may be used to analyze tracebacks. As of version 0.11.0, a pin number is written to the log when the server is started:

```
INFO werkzeug:87  * Debugger pin code: 123-456-789
```

The pin code must be entered once per debugging session. If you will never use the built-in server for public access, you may disable the pin check by adding:

```
WERKZEUG_DEBUG_PIN=off
```

to your OS's environment variables. See Werkzeug docs for more information.

Using the built-in server for production

Caution: Using the built-in server for public wikis is not recommended. Should you wish to do so, turn off the werkzeug debugger and auto reloader by passing the -d and -r flags. The wikiconfig.py settings of *DEBUG* = *False* and *TESTING* = *False* are ignored by the built-in server. You must use the -d and -r flags. See Werkzeug docs for more information.:

moin moin –host 0.0.0.0 –port 80 -D -R

5.4.2 External Web Server (advanced)

We won't go into details about using moin under an external web server, because every web server software is different and has its own documentation, so please read the documentation that comes with it. Also, in general, server administration requires advanced experience with the operating system, permissions management, dealing with security, the server software, etc.

In order to use MoinMoin with another web server, ensure that your web server can talk to the moin WSGI application, which you can get using this code:

```
from MoinMoin.app import create_app
application = create_app('/path/to/config/wikiconfig.py')
```

MoinMoin is a Flask application, which is a micro framework for WSGI applications, so we recommend you read Flask's good deployment documentation.

Make sure you use *create_app()* as shown above to create the application, because you can't import the application from MoinMoin.

Continue reading here: http://flask.pocoo.org/docs/deploying/

In case you run into trouble with deployment of the moin WSGI application, you can try a simpler WSGI app first. See *docs/examples/deployment/test.wsgi*.

As long as you can't make *test.wsgi* work, the problem is not with moin, but rather with your web server and WSGI app deployment method.

When the test app starts doing something other than Server Error 500, please proceed with the MoinMoin app and its configuration. Otherwise, read your web server error log files to troubleshoot the issue from there.

Tip: Check contents of /contrib/wsgi/ for sample wsgi files for your server.

5.4.3 Create and Serve a Static Wiki Image

"dump-html" is a utility used to create static html dumps of MoinMoin wiki content. You may find it useful to create a static dump for a software release, a high volume read-only copy for a busy web site, or a thumb drive version to carry on trips when you do not have internet access.

To execute dump-html, use the command line interface. The following three commands are equivalent as the specified options are the defaults.

```
moin dump-html
moin dump-html --directoy HTML --theme topside_cms --exclude-ns userprofiles --query .
↪*
moin dump-html -d HTML -t topside_cms -e userprofiles -q .*
```

The –directory option may be a relative or absolute path. The default directory, HTML, will be placed under the wiki root.

The –theme option specifies the theme. See "Customize the CMS Theme" within the "Introduction into MoinMoin Configuration" section for alternatives.

The –exclude-ns option specifies a comma separated list of namespaces that will be excluded from the dump. The "userprofiles" namespace should always be excluded. To exclude user home pages from the static dump, use **userprofiles,users** with no embedded spaces.

The –query option may be a single page name or a regex selecting the items to be included in the dump. The default of ".*" selects all items.

Once created, the HTML directory may be moved anywhere as all the internal links are relative. The pages may be served using your favorite web server or directly from the file system.

Warning: Some browsers (Chrome, IE11, Opera) serve files loaded from the OS file system as plain text. https://github.com/moinwiki/moin/issues/641

5.5 Introduction into MoinMoin Configuration

5.5.1 Kinds of configuration files

To change how moinmoin behaves and looks, you may customize it by editing its configuration files:

- Wiki Engine Configuration
  - the file is often called wikiconfig.py, but it can have any name
  - in that file, there is a Config class; this is the wiki engine's configuration
  - it is written in Python
- Framework Configuration
  - this is also located in the same file as the Wiki Engine Configuration
  - there are some UPPERCASE settings at the bottom; this is the framework's config (for Flask and Flask extensions)
  - it is written in Python
- Logging Configuration
  - optional; if you don't configure this, it will use the builtin defaults
  - this is a separate file, often called logging.conf
  - it has an .ini-like file format

Do small steps and have backups

Start from one of the sample configs provided with moin and only perform small changes, then try it before testing the next change.

If you're not used to the config file format, backup your last working config so you can revert to it in case you make some hard to find typo or other error.

Editing Python files

When editing Python files, be careful with indentation, only use multiples of 4 spaces to indent, and no tabs!

Also, be careful with syntax in general, it must be valid Python code or else it will crash with some error when trying to load the config. If that happens, read the error message, it will usually tell the line number and what the problem is. If you can't fix it easily, then revert to your backup of your last working config.

Why use Python for configuration?

At first, you might wonder why we use Python code for configuration. One of the reasons is that it is a powerful language. MoinMoin itself is developed in Python and using something else would usually mean much more work when developing new functionality.

5.5.2 Directory Structure

Shown below are parts of the directory structure after cloning moin and running quickinstall.py. The default uses the OS file system for storage of wiki data and indexes. The directories and files shown are referenced in this section of documentation related to configuration:

```
moin/                       # clone root, default name
    contrib/                # scripts and docs of interest to developers
    docs/                   # moin documentation in restructured text (.rst) format
        _build/
            html/           # local copy of moin documentation, created by running
↪"./m docs" command
    requirements.d/         # package requirements used by quickinstall.py
    scripts/                # misc. scripts of interest to developers
    src/
        moin/               # large directory containing moin application code
    wiki/                   # the wiki instance; created by running "./m sample" or
↪"./m new-wiki" commands
        data/               # wiki data and metadata
        index/              # wiki indexes
    wiki_local/             # a convenient location to store custom CSS, Javascript,
↪ templates, logos, etc.
    wikiconfig.py           # main configuration file, modify this to add or change␣
↪features
    intermap.txt            # interwiki map: copied by quickinstall.py, updated by
↪"./m interwiki"
```

After installing moin from pypi or unpacking using a package manager, the directory structure will look like this:

```
myvenv/                 # virtualenv root
    bin/                # Windows calls this directory Scripts
    include             # Windows calls this directory Include
    lib/                # Windows calls this directory Lib, includes moin package
```

After activating the above venv, *moin create-instance -p <mywiki>* creates the structure below. Multiple instances of *mywiki* can be created with different names. *mywiki* may be created as a subdirectory of *myvenv* or elsewhere. The *preview* and *sql* subdirectories are created when a user edits a wiki item. To run moin using the built-in server, cd to the <*mywiki*> directory and execute *moin run*.:

```
mywiki/                 # wikiconfig dir, use this as CWD for moin commands
    wiki/               # the wiki instance; created by `moin create-instance`
        data/           # wiki data and metadata
        index/          # wiki indexes
        preview/        # text item backups are created when user clicks edit Preview␣
↪button
        sql/            # sqlite database used for edit locking
    wiki-local/         # store custom CSS, Javascript, templates, logos, etc. here
    wikiconfig.py       # main configuration file, modify this to add or change␣
↪features
    intermap.txt        # list of external wikis used in wikilinks:␣
↪[[MeatBall:InterWiki]]
```

5.5.3 wikiconfig.py Layout

A wikiconfig.py looks like this:

```
# -*- coding: utf-8 -*-
from moin.config.default import DefaultConfig

class Config(DefaultConfig):
    # some comment
    sometext = 'your value'
    somelist = [1, 2, 3]

MOINCFG = Config  # Flask only likes uppercase characters
SOMETHING_FLASKY = 'foobar'
```

Let's go through this line-by-line:

0. this declares the encoding of the config file; make sure your editor uses the same encoding (character set), especially if you intend to use non-ASCII characters.
1. this gets the DefaultConfig class from the moin code; it has default values for all settings and this will save you work, because you only have to define the parts that should be different from the defaults.
2. empty line, for better readability
3. define a new class *Config* that inherits most content from *DefaultConfig*; this is the wiki engine configuration and if you define some setting within this class, it will overwrite the setting from DefaultConfig.
4. a # character defines a comment in your config. This line, as well as all other following lines with Config settings, is indented by 4 blanks, because Python defines blocks by indentation.
5. define a Config attribute called *sometext* with value 'your value'.
6. define a Config attribute called *somelist* with value [1, 2, 3]; this is a list with the numbers 1, 2 and 3 as its elements.
7. empty line, for better readability
8. the special line "MOINCFG = Config" must stay there in exactly this form for technical reasons.
9. UPPERCASE code at the bottom, outside the Config class is a framework configuration; usually something for Flask or some Flask extension.

A real-life example of a *wikiconfig.py* can be found in the *docs/examples/config/* directory.

5.6 Wiki Engine Configuration

5.6.1 User Interface Customization

Customizing a wiki usually requires adding a few files that contain custom templates, logo image, CSS, etc. To accomplish this, a directory named "wiki_local" is provided. One advantage of using this directory and following the examples below is that MoinMoin will serve the files.

If desired, the name of this directory may be changed or a separate subdirectory for template files may be created by editing the wikiconfig file and changing the line that defines *template_dirs*:

```
template_dirs = [os.path.join(wikiconfig_dir, 'wiki_local'), ]
```

Using a custom snippets.html template

The user interface or html elements that often need customization are defined as macros in the template file *snippets.html*.

If you would like to customize some parts, you have to copy the built-in *src/moin/templates/snippets.html* file and save it in the *wiki_local* directory so moin can use your copy instead of the built-in one.

To customize something, you usually have to insert your code between the *{% macro ... %}* and *{% endmacro %}* lines, see below for more details.

Logo

To replace the default MoinMoin logo with your own logo, copy your logo to *wiki_local* and change the logo macro to something like:

```
{% macro logo() -%}
    <img src="{{ url_for('serve.files', name='wiki_local', filename='MyLogo.png') }}"␣
↪id="moin-img-logo" alt="Logo">
{%- endmacro %}
```

This is recommended to allow your users to immediately recognize which wiki site they are currently on.

You can use text or even nothing at all for the logo, it is not required to be an image:

```
{% macro logo() -%}
    <span style="font-size: 50px; color: red;">My Wiki</span>
{%- endmacro %}
```

Make sure the dimensions of your logo image or text fit into the layout of the theme(s) your wiki users are using.

Displaying license information

If you need to display something like license information for your content or some other legalese, use this macro:

```
{# License information in the footer #}
{% macro license_info() -%}
All wiki content is licensed under the WTFPL.
{%- endmacro %}
```

Inserting pieces of HTML

At some specific places, you can add a piece of your own html into the head or body of the theme's html output:

```
{# Additional HTML tags inside <head> #}
{% macro head() -%}
{%- endmacro %}

{# Additional HTML before #moin-header #}
{% macro before_header() -%}
{%- endmacro %}

{# Additional HTML after #moin-header #}
{% macro after_header() -%}
```

(continues on next page)

(continued from previous page)

```
{%- endmacro %}

{# Additional HTML before #moin-footer #}
{% macro before_footer() -%}
{%- endmacro %}

{# Additional HTML after #moin-footer #}
{% macro after_footer() -%}
{%- endmacro %}
```

Credits and Credit Logos

At the bottom of your wiki pages, usually some text and image links are shown pointing out that the wiki runs MoinMoin, uses Python, that MoinMoin is GPL licensed, etc.

If you run a public site using MoinMoin, we would appreciate if you *keep* those links, especially the "MoinMoin powered" one.

However, if you can't do that for some reason, feel free to modify these macros to show something else:

```
{# Image links in the footer #}
{% macro creditlogos(start='<ul id="moin-creditlogos"><li>'|safe, end='</li></ul>
↪'|safe, sep='</li><li>'|safe) %}
{{ start }}
{{ creditlogo('https://moinmo.in/', url_for('.static', filename='logos/moinmoin_
↪powered.png'),
   'MoinMoin powered', 'This site uses the MoinMoin Wiki software.') }}
{{ sep }}
{{ creditlogo('https://moinmo.in/Python', url_for('.static', filename='logos/python_
↪powered.png'),
   'Python powered', 'MoinMoin is written in Python.') }}
{{ end }}
{% endmacro %}

{# Text links in the footer #}
{% macro credits(start='<p id="moin-credits">'|safe, end='</p>'|safe, sep='<span>&
↪bull;</span>'|safe) %}
{{ start }}
{{ credit('https://moinmo.in/', 'MoinMoin Powered', 'This site uses the MoinMoin Wiki␣
↪software.') }}
{{ sep }}
{{ credit('https://moinmo.in/Python', 'Python Powered', 'MoinMoin is written in␣
↪Python.') }}
{{ sep }}
{{ credit('https://moinmo.in/GPL', 'GPL licensed', 'MoinMoin is GPL licensed.') }}
{{ sep }}
{{ credit('http://validator.w3.org/check?uri=referer', 'Valid HTML 5', 'Click here to␣
↪validate this page.') }}
{{ end }}
{% endmacro %}
```

Adding scripts

You can add scripts like this:

```
{# Additional Javascript #}
{% macro scripts() -%}
<script type="text/javascript" src="{{ url_for('serve.files', name='wiki_local',␣
↪filename='MyScript.js') }}"></script>
{% endmacro %}
```

Adding CSS

To apply some style changes, add some custom css and overwrite any style you don't like in the base theme:

```
{# Additional Stylesheets (after theme css, before user css #}
{% macro stylesheets() -%}
    <link media="screen" href="{{ url_for('serve.files', name='wiki_local', filename=
↪'company.css') }}" title="Company CSS" rel="stylesheet" />
    <link media="screen" href="{{ url_for('serve.files', name='wiki_local', filename=
↪'red.css') }}" title="Red Style" rel="alternate stylesheet" />
    <link media="screen" href="{{ url_for('serve.files', name='wiki_local', filename=
↪'green.css') }}" title="Green Style" rel="alternate stylesheet" />
{%- endmacro %}
```

You can either add some normal css stylesheet or add a choice of alternate stylesheets.

See:

- CSS media types
- Alternate Stylesheets

A good way to test a stylesheet is to first use it as user CSS before configuring it for the public.

Please note that *stylesheets* will be included no matter what theme the user has selected, so either only apply changes to all available themes or force all users to use the same theme, so that your CSS displays correctly.

Customize the CMS Theme

Moin provides one CMS theme: the Topside CMS Theme.

The CMS theme replaces the wiki navigation links used by editors and administrators with a few links to the most important items within your wiki. Wiki admins may want to make the CMS theme the default theme when:

- Casual visitors are interested in viewing the wiki content, but confused by the wiki navigation links.
- Errant bots are overloading your server by following the wiki navigation links on every page.
- Contributors do not mind logging in before editing.

Customizing the CMS header may be done as follows. Several restarts of the server may be required:

- Copy /templates/snippets.html to the wiki_local directory and find the *macro cms_header*.
- Usually the logo, sitename, and search form sections are not changed.
- If a link to login is wanted, leave the "request.user_agent" section as is, else remove the entire block.
- Add or remove links in the navibar section as required, defaults links include Home page and Global Index.
- If many links are desired, consider using *macro custom_panels*.
- Test by logging in and setting "Topside CMS Theme" as your preferred theme.

- After testing, make the cms theme the default theme by adding `theme_default = "topside_cms"` to wikiconfig.
- Inform your editors to login and set another theme as their preferred theme.
- If the login link was removed, the login page is available by keying `+login` as the page name in the browser URL.

Here is the source code segment from snippets.html:

```
{# Header/Sidebar for topside_cms theme - see docs for tips on customization #}
{% macro cms_header() %}
    <header id="moin-header">
        {% block header %}

            {% if logo() %}
                <div id="moin-logo">
                    <a href="{{ url_for('frontend.show_item', item_name=cfg.root_
↪mapping.get('', cfg.default_root)) }}">
                        {{ logo() }}
                    </a>
                </div>
            {%- endif %}

            {% if cfg.sitename %}
                <a class="moin-sitename" href="{{ url_for('frontend.show_item', item_
↪name=cfg.root_mapping.get('', cfg.default_root)) }}">
                    {{ cfg.sitename }}
                </a>
                <br>
            {%- endif %}

            {% if request.user_agent and search_form %} {# request.user_agent is true␣
↪if browser, false if run as moin dump-html #}
                {{ utils.header_search(search_form) }}
            {% endif %}

            {% if request.user_agent %} {# request.user_agent is true if browser,␣
↪false if run as moin dump-html #}
                <ul id="moin-username" class="moin-header-links">
                    {{ utils.user_login_logoff() }}
                </ul>
            {%- endif %}

            <ul id="moin-navibar" class="moin-header-links panel">
                {# wiki admins should add links and headings for key items within the␣
↪local wiki below #}
                <li class="moin-panel-heading">Navigation</li>
                <li class="wikilink"><a href="{{ url_for('frontend.show_item', item_
↪name='Home') }}">Start</a></li>
                <li class="wikilink"><a href="{{ url_for('frontend.show_item', item_
↪name='+index') }}">Index</a></li>
            </ul>

            {{ custom_panels() }}

        {% endblock %}
    </header>
    <br>
```

(continues on next page)

(continued from previous page)

```
{% endmacro %}
```

Displaying user avatars

Optionally, moin can display avatar images for the users, using gravatar.com service. To enable it, add or uncomment this line in wikiconfig:

```
user_use_gravatar = True
```

Please note that using the gravatar service has some privacy issues:

- to register your image for your email at gravatar.com, you need to give them your email address, which is the same as you use in your wiki user profile.
- when the wiki displays an avatar image on some item / view, the URL will be exposed as referrer to the avatar service provider, so they will roughly know which people read or work on which wiki items / views.

XStatic Packages

XStatic is a packaging standard to package external static files as a Python package, often third party. That way they are easily usable on all operating systems, whether it has a package management system or not.

In many cases, those external static files are maintained by someone else (like jQuery javascript library or larger js libraries) and we definitely do not want to merge them into our project.

For MoinMoin we require the following XStatic Packages in setup.py:

- jquery for jquery lib functions loaded in the template file base.html
- jquery_file_upload loaded in the template file of index view. It allows to upload many files at once.
- bootstrap used by the basic theme.
- font_awesome provides text icons.
- ckeditor used in template file modify_text_html. A WYSIWYG editor similar to word processing desktop editing applications.
- autosize used by basic theme to adjust textarea on modify view.
- svgedit_moin is loaded at template modify_svg-edit. It is a fast, web-based, Javascript-driven SVG editor.
- twikidraw_moin a Java applet loaded from template file of modify_twikidraw. It is a simple drawing editor.
- anywikidraw a Java applet loaded from template file of modify_anywikidraw. It can be used for editing drawings and diagrams on items.
- jquery_tablesorter used to provide client side table sorting.
- pygments used to style code fragments.

These packages are imported in wikiconfig by:

```
from xstatic.main import XStatic
# names below must be package names
mod_names = [
    'jquery', 'jquery_file_upload',
    'bootstrap',
    'font_awesome',
```

(continues on next page)

(continued from previous page)

```
    'ckeditor',
    'autosize',
    'svgedit_moin', 'twikidraw_moin', 'anywikidraw',
    'jquery_tablesorter',
    'pygments',
]
pkg = __import__('xstatic.pkg', fromlist=mod_names)
for mod_name in mod_names:
    mod = getattr(pkg, mod_name)
    xs = XStatic(mod, root_url='/static', provider='local', protocol='http')
    serve_files[xs.name] = xs.base_dir
```

In a template file you access the files of such a package by its module name:

```
url_for('serve.files', name='the mod name', filename='the file to load')
```

Adding XStatic Packages

The following example shows how you can enable the additional package XStatic-MathJax which is used for mathml or latex formulas in an item's content.

- install xstatic-mathjax (e.g. using `pip install xstatic-mathjax`)
- add the name 'mathjax' to to the list of mod_names in wikiconfig
- copy /templates/snippets.html to the wiki_local directory
- modify the snippets.html copy by adding the required fragment to the scripts macro:

```
{% macro scripts() -%}
<script type="text/x-mathjax-config">
MathJax.Hub.Config({
    extensions: ["tex2jax.js"],
    jax: ["input/TeX","output/HTML-CSS"],
    tex2jax: {inlineMath: [["$","$"],["\\(","\\)"]]}
});
</script>
<script src="{{ url_for('serve.files', name='mathjax', filename='MathJax.js') }}">
↪</script>
{%- endmacro %}
```

Custom Themes

In case you want to do major changes to how MoinMoin displays its pages, you could also write your own theme.

Caution: developing your own theme means you also have to maintain and update it, which normally requires a long-term effort.

To add a new theme, add a new directory under src/moin/themes/ where the directory name is the name of your theme. Note the directory structure under the other existing themes. Copy an *info.json* file to your theme directory and edit as needed. Create a file named theme.css in the src/moin/themes/<theme name>/static/css/ directory.

To change the layout of the theme header, sidebar and footer, create a templates/ directory and copy and modify the files layout.html and show.html from either src/moin/templates/ or one of the existing theme templates directories.

For many themes, modifying the files noted above will be sufficient. If changes to views are required, copy additional template files. If there is a requirement to change the MoinMoin base code, please consider submitting a patch.

5.6.2 Authentication

MoinMoin uses a configurable *auth* list of authenticators, so the admin can configure whatever he/she likes to allow for authentication. Moin processes this list from left to right.

Each authenticator is an instance of some specific class, configuration of the authenticators usually works by giving them keyword arguments. Most have reasonable defaults though.

MoinAuth

This is the default authentication moin uses if you don't configure something else. The user logs in by filling out the login form with username and password, moin compares the password hash against the one stored in the user's profile and if both match, the user is authenticated:

```
from moin.auth import MoinAuth
auth = [MoinAuth()]  # this is the default!
```

HTTPAuthMoin

With HTTPAuthMoin moin does http basic authentication by itself without the help of the web server:

```
from moin.auth.http import HTTPAuthMoin
auth = [HTTPAuthMoin(autocreate=True)]
```

If configured like that, moin will request authentication by emitting a http header. Browsers then usually show some login dialogue to the user, asking for username and password. Both then gets transmitted to moin and it is compared against the password hash stored in the user's profile.

Note: when HTTPAuthMoin is used, the browser will show that login dialogue, so users must login to use the wiki.

GivenAuth

With GivenAuth moin relies on the webserver doing the authentication and giving the result to moin, usually via the environment variable REMOTE_USER:

```
from moin.auth import GivenAuth
auth = [GivenAuth(autocreate=True, coding='utf-8')]
```

Using this method has some pros and cons:

- you can use lots of authentication extensions available for your web server
- but the only information moin will get via REMOTE_USER is the authenticated user's name, nothing else. So, e.g. for LDAP/AD, you won't get additional content stored in the LDAP directory.
- everything you won't get, but which you need, will need to be manually stored and updated in the user's profile, e.g. the user's email address, etc.

Please note that you must give the correct character set so that moin can decode the username to unicode, if necessary. For environment variables like REMOTE_USER, the coding might depend on your operating system.

If you do not know the correct coding, try: 'utf-8', 'iso-8859-1', ...

Todo: add the usual coding(s) for some platforms (like windows)

To try it out, change configuration, restart moin and then use some non-ASCII username (like with german umlauts or accented characters). If moin does not crash (log a Unicode Error), you have likely found the correct coding.

For users configuring GivenAuth on Apache, an example virtual host configuration file is included with MoinMoin in *docs/examples/deployment/moin-http-basic-auth.conf*.

LDAPAuth

With LDAPAuth you can authenticate users against a LDAP directory or MS Active Directory service.

LDAPAuth with single LDAP server

This example shows how to use LDAPAuth with a single LDAP/AD server:

```
from moin.auth.ldap_login import LDAPAuth
ldap_common_arguments = dict(
    # the values shown below are the DEFAULT values (you may remove them if you are␣
↪happy with them),
    # the examples shown in the comments are typical for Active Directory (AD) or␣
↪OpenLDAP.
    bind_dn='',  # We can either use some fixed user and password for binding to LDAP.
                 # Be careful if you need a % char in those strings - as they are␣
↪used as
                 # a format string, you have to write %% to get a single % in the end.
                 #bind_dn = 'binduser@example.org' # (AD)
                 #bind_dn = 'cn=admin,dc=example,dc=org' # (OpenLDAP)
                 #bind_pw = 'secret'
                 # or we can use the username and password we got from the user:
                 #bind_dn = '%(username)s@example.org' # DN we use for first bind (AD)
                 #bind_pw = '%(password)s' # password we use for first bind
                 # or we can bind anonymously (if that is supported by your␣
↪directory).
                 # In any case, bind_dn and bind_pw must be defined.
    bind_pw='',
    base_dn='',  # base DN we use for searching
                 #base_dn = 'ou=SOMEUNIT,dc=example,dc=org'
    scope=2, # scope of the search we do (2 == ldap.SCOPE_SUBTREE)
    referrals=0, # LDAP REFERRALS (0 needed for AD)
    search_filter='(uid=%(username)s)',  # ldap filter used for searching:
                                         #search_filter = '(sAMAccountName=
↪%(username)s)' # (AD)
                                         #search_filter = '(uid=%(username)s)' #␣
↪(OpenLDAP)
                                         # you can also do more complex filtering␣
↪like:
                                         # "(&(cn=%(username)s)(memberOf=CN=WikiUsers,
↪OU=Groups,DC=example,DC=org))"
    # some attribute names we use to extract information from LDAP (if not None,
    # if None, the attribute won't be extracted from LDAP):
    givenname_attribute=None, # often 'givenName' - ldap attribute we get the first␣
↪name from
    surname_attribute=None, # often 'sn' - ldap attribute we get the family name from
    aliasname_attribute=None, # often 'displayName' - ldap attribute we get the␣
↪aliasname from
    email_attribute=None, # often 'mail' - ldap attribute we get the email address␣
↪from
```

(continues on next page)

(continued from previous page)

```
    email_callback=None, # callback function called to make up email address
    coding='utf-8', # coding used for ldap queries and result values
    timeout=10, # how long we wait for the ldap server [s]
    start_tls=0, # usage of Transport Layer Security 0 = No, 1 = Try, 2 = Required
    tls_cacertdir=None,
    tls_cacertfile=None,
    tls_certfile=None,
    tls_keyfile=None,
    tls_require_cert=0, # 0 == ldap.OPT_X_TLS_NEVER (needed for self-signed certs)
    bind_once=False, # set to True to only do one bind - useful if configured to bind␣
↪as the user on the first attempt
    autocreate=True, # set to True to automatically create/update user profiles
    report_invalid_credentials=True, # whether to emit "invalid username or password"␣
↪msg at login time or not
)

ldap_authenticator1 = LDAPAuth(
    server_uri='ldap://localhost',  # ldap / active directory server URI
                                    # use ldaps://server:636 url for ldaps,
                                    # use  ldap://server for ldap without tls (and␣
↪set start_tls to 0),
                                    # use  ldap://server for ldap with tls (and set␣
↪start_tls to 1 or 2).
    name='ldap1',  # unique name for the ldap server, e.g. 'ldap_pdc' and 'ldap_bdc'␣
↪(or 'ldap1' and 'ldap2')
    **ldap_common_arguments  # expand the common arguments
)

auth = [ldap_authenticator1, ] # this is a list, you may have multiple ldap␣
↪authenticators
                               # as well as other authenticators

# customize user preferences (optional, see MoinMoin/config/multiconfig for internal␣
↪defaults)
# you maybe want to use user_checkbox_remove, user_checkbox_defaults, user_form_
↪defaults,
# user_form_disable, user_form_remove.
```

LDAPAuth with two LDAP servers

This example shows how to use LDAPAuth with a two LDAP/AD servers, such as in a setup with a primary controller and backup domain controller:

```
# ... same as for single server (except the line with "auth =") ...
ldap_authenticator2 = LDAPAuth(
    server_uri='ldap://otherldap',  # ldap / active directory server URI for second␣
↪server
    name='ldap2',
    **ldap_common_arguments
)

auth = [ldap_authenticator1, ldap_authenticator2, ]
```

AuthLog

AuthLog is not a real authenticator in the sense that it authenticates (logs in) or deauthenticates (logs out) users. It is passively logging informations for authentication debugging:

```
from moin.auth import MoinAuth
from moin.auth.log import AuthLog
auth = [MoinAuth(), AuthLog(), ]
```

Example logging output:

```
2011-02-05 16:35:00,229 INFO MoinMoin.auth.log:22 login: user_obj=<moin.user.User at␣
↪0x90a0f0c name:'ThomasWaldmann' valid:1> kw={'username': 'ThomasWaldmann', 'attended
↪': True, 'multistage': None, 'login_password': 'secret', 'login_username':
↪'ThomasWaldmann', 'password': 'secret', 'login_submit': ''}
2011-02-05 16:35:04,716 INFO MoinMoin.auth.log:22 session: user_obj=<MoinMoin.user.
↪User at 0x90a0f6c name:'ThomasWaldmann' valid:1> kw={}
2011-02-05 16:35:06,294 INFO MoinMoin.auth.log:22 logout: user_obj=<MoinMoin.user.
↪User at 0x92b5d4c name:'ThomasWaldmann' valid:False> kw={}
2011-02-05 16:35:06,328 INFO MoinMoin.auth.log:22 session: user_obj=None kw={}
```

Note: there is sensitive information like usernames and passwords in this log output. Make sure you only use this for testing only and delete the logs when done.

SMBMount

SMBMount is no real authenticator in the sense that it authenticates (logs in) or deauthenticates (logs out) users. It instead catches the username and password and uses them to mount a SMB share as this user.

SMBMount is only useful for very special applications, e.g. in combination with the fileserver storage backend:

```
from moin.auth.smb_mount import SMBMount

smbmounter = SMBMount(
    # you may remove default values if you are happy with them
    # see man mount.cifs for details
    server='smb.example.org',  # (no default) mount.cifs //server/share
    share='FILESHARE',  # (no default) mount.cifs //server/share
    mountpoint_fn=lambda username: '/mnt/wiki/%s' % username,  # (no default)␣
↪function of username to determine the mountpoint
    dir_user='www-data',  # (no default) username to get the uid that is used for␣
↪mount.cifs -o uid=...
    domain='DOMAIN',  # (no default) mount.cifs -o domain=...
    dir_mode='0700',  # (default) mount.cifs -o dir_mode=...
    file_mode='0600',  # (default) mount.cifs -o file_mode=...
    iocharset='utf-8',  # (default) mount.cifs -o iocharset=... (try 'iso8859-1' if␣
↪default does not work)
    coding='utf-8',  # (default) encoding used for username/password/cmdline (try
↪'iso8859-1' if default does not work)
    log='/dev/null',  # (default) logfile for mount.cifs output
)

auth = [....., smbmounter]  # you need a real auth object in the list before␣
↪smbmounter

smb_display_prefix = "S:"  # where //server/share is usually mounted for your windows␣
↪users (display purposes only)
```

Todo: check if SMBMount still works as documented

5.6.3 Transmission security

Credentials

Some of the authentication methods described above will transmit credentials, like usernames and password, in unencrypted form:

- MoinAuth: when the login form contents are transmitted to moin, they contain username and password in clear text.
- HTTPAuthMoin: your browser will transfer username and password in a encoded (but NOT encrypted) form with EVERY request; it uses http basic auth.
- GivenAuth: check the potential security issues of the authentication method used by your web server; for http basic auth please see HTTPAuthMoin.

Contents

http transmits everything in clear text and is therefore not encrypted.

Encryption

Transmitting unencrypted credentials or contents can cause serious issues in many scenarios.

We recommend you make sure the connections are encrypted, like with https or VPN or an ssh tunnel.

For public wikis with very low security / privacy needs, it might not be needed to encrypt the content transmissions, but there is still an issue for the credential transmissions.

When using unencrypted connections, wiki users are advised to make sure they use unique credentials and not reuse passwords that are used for other purposes.

5.6.4 Password security

Password strength

As you might know, many users are bad at choosing reasonable passwords and some are tempted to use easily crackable passwords.

To help users choose reasonable passwords, moin has a simple builtin password checker that is enabled by default and does some sanity checks, so users don't choose easily crackable passwords.

It **does** check:

- length of password (default minimum: 8)
- amount of different characters in password (default minimum: 5)
- password does not contain user name
- user name does not contain password

- password is not a keyboard sequence (like "ASDFghjkl" or "987654321"), currently we have only US and DE keyboard data built-in.

It **does not** check:

- whether the password is in a well-known dictionary or password list
- whether a password cracker can break it

If you are not satisfied with the default values, you can easily customize the checker:

```
from moin.config.default import DefaultConfig, _default_password_checker
password_checker = lambda cfg, name, pw: _default_password_checker(
                        cfg, name, pw, min_length=10, min_different=6)
```

You could also completely replace it with your own implementation.

If your site has rather low security requirements, you can disable the checker by:

```
password_checker = None  # no password checking
```

Password storage

Moin never stores wiki user passwords in clear text, but uses strong cryptographic hashes provided by the "passlib" library, see there for details:

http://packages.python.org/passlib/.

The passlib docs recommend 3 hashing schemes that have good security: sha512_crypt, pbkdf2_sha512 and bcrypt (bcrypt has additional binary/compiled package requirements, please refer to the passlib docs in case you want to use it).

By default, we use sha512_crypt hashes with default parameters as provided by passlib (this is same algorithm as moin >= 1.9.7 used by default).

In case you experience slow logins or feel that you might need to tweak the hash generation for other reasons, please read the passlib docs. moin allows you to configure passlib's CryptContext params within the wiki config, the default is this:

```
passlib_crypt_context = dict(
    schemes=["sha512_crypt", ],
)
```

5.6.5 Authorization

Moin uses Access Control Lists (ACLs) to specify who is authorized to perform a given action. ACLs enable wiki administrators and possibly users to choose between *soft security* and *hard security*.

- if your wiki is rather open (soft security), you make it easy to contribute, e.g. a user who is not a regular user of your wiki could fix some typos he has just found. However, a hostile user or bot could easily add spam into your wiki. In this case, an active user community can quickly detect and remove the spam.
- if your wiki is rather closed (hard security), e.g. you require every user to first apply for an account and to log in before being able to do changes, you will rarely get contributions from casual users and possibly discourage contributions from members of your community. But, getting spam is then less likely.
- ACLs provide the means of using both methods. Key wiki items that are frequently viewed and infrequently changed may be updated only by selected users while other items that are frequently changed may be updated by any user.

Moin's default configuration makes use of *soft security* which is in use by many wikis to maximize collaboration among its user community.

Wiki administrators may harden security by reconfiguring the default ACLs. Later, as wiki items are created and updated, the default configuration may be overridden by setting an ACL on the item.

Hardening security implies that there will be a registration and login process that enables individual users to gain privileges. While wikis with a small user community may function with ACLs specifying only usernames, larger wikis will make use of ACLs that reference groups or lists of usernames. The definitions of built-in groups and creation of groups are discussed below under the headings *ACLs - special groups* and *Groups*.

ACL for functions

Moin has some built in functions that are protected by ACLs:

- superuser - used for miscellaneous administrative functions. Give this only to highly trusted people

Example:

```
acl_functions = 'YourName:superuser'
```

ACLs for contents

This type of ACL controls access to content stored in the wiki. Wiki items may have ACLs defined in their metadata. Within wikiconfig, ACLs are specified per namespace and storage backend (see storage backend docs for details). The example below shows an entry for the default namespace:

```
default_acl=dict(before='SuperUser:read,write,create,destroy,admin',
                 default='TrustedEditorGroup:read,write,create,destroy,admin␣
↪Known:read,write,create',
                 after='All:read',
                 hierarchic=False, ),
```

As shown above, *before*, *default* and *after* ACLs are specified. The *default* ACL is only used if no ACL is specified in the metadata of the target item.

How to use before, default, and after:

- *before* is usually used to force something, for example if you want to give some wiki admin all permissions indiscriminately; in the example above, no one can create an item ACL rule locking out SuperUser's access
- *default* is the behavior if no ACL was created in the item's metadata; above, only members of a trusted group can write ACL rules or delete items, and a user must be logged in (known) to write or create items
- *after* is rarely used and when it is, it is used to "not forget something unless otherwise specified"; above, all users may read all items unless blocked (or given more privileges) by an ACL on the target item

When configuring content ACLs, you can choose between standard (flat) ACL processing and hierarchic ACL processing. Hierarchic processing means that subitems inherit ACLs from their parent items if they don't have an ACL themselves.

Note that while hierarchic ACLs are rather convenient sometimes, they make the system more complex. You have to be very careful with permission changes happening as a result of changes in the hierarchy, such as when you create, rename or delete items. When multiple item names are used the complexity increases even more because all parents are searched for ACLs – if conflicting allow/deny ACLs are found allow always wins.

Supported capabilities (rights):

- read - read content
- write - write (edit, modify, delete) content
- create - create new items
- destroy - completely destroy revisions or items; to be given only to *fully-trusted* users
- admin - change (create, remove) ACLs for the item; to be given only to *fully-trusted* users

The write capability includes the authority to delete an item since any user with write authority may edit and remove or replace all content. A deleted item does not appear in the Global Index, but the deletion event does appear in the global history. To recover a deleted item, find the deleted item line in global history, click the link to the item's history, and then click a revert link to one of the prior revisions.

ACLs - special groups

In addition to the groups provided by the group backend(s), there are some special group names available within ACLs. These names are case-sensitive and must be capitalized as shown:

- All - a virtual group containing every user
- Known - a virtual group containing every logged-in user
- Trusted - a virtual group containing every logged-in user who was logged in by some specific "trusted" authentication method

ACLs - basic syntax

An ACL is a unicode string with one or more access control entries which are space separated.

An entry is a colon-separated set of two values:

- the left side is a comma-separated list of user and/or group names
- the right side is a comma-separated list of rights / capabilities for those users/groups.

An ACL is processed from left to right, where the first left-side match counts.

Example:

```
"SuperMan,WonderWoman:read,write,create,destroy,admin All:read,write"
```

If "SuperMan" is currently logged in and moin processes this ACL, it will find a name match in the first entry. If moin wants to know whether he may destroy, the answer will be "yes", as destroy is one of the capabilities/rights listed on the right side of this entry.

If "JoeDoe" is currently logged in and moin processes this ACL, the first entry won't match, so moin will proceed left-to-right and look at the second entry. Here we have the special group name, "All" (and JoeDoe is obviously a member of this group), so this entry matches. If moin wants to know whether he may destroy, the answer will be "no", as destroy is not listed on the right side of the "All" entry. If moin wants to know whether he may write, the answer will be "yes".

Notes:

- As a consequence of the left-to-right and first-match-counts processing, you must order ACL entries so that the more specific ones (like for "SuperMan") are left of the less specific ones. Usually, you want this order:

  1) usernames

  2) special groups

3) more general groups
4) Trusted
5) Known
6) All

- Do not put any spaces into an ACL entry, unless it is part of a user or group name.
- A right that is not explicitly given by an applicable ACL is denied.

ACLs - entry prefixes

To make the system more flexible, there are two ways to modify an ACL entry: prefixing it with a '+' or a '-'.

If you use one of the two, MoinMoin will search for both a username and permission, and a match will have to match both the name of user (left-side) *and* the permission MoinMoin is searching for (right-side), otherwise it will continue with the next entry.

'+' indicates that MoinMoin should give the permission(s) specified on the right side.

'-' indicates that MoinMoin should deny the permission(s) specified on the right side.

Example:

```
"+SuperMan:create,destroy,admin -Idiot:write All:read,write"
```

If "SuperMan" is currently logged in and moin wants to know whether he may destroy, it'll find a match in the first entry, because the name matches *and* permission in question matches. As the prefix is '+', the answer is "yes". If moin wants to know whether he may write, the first entry will not match on both sides, so moin will proceed and look at the second entry. It doesn't match, so it will look at the third entry. Of course "SuperMan" is a member of group "All", so we have a match here. As "write" is listed on the right side, the answer will be "yes".

If "Idiot" is currently logged in and moin wants to know whether he may write, it will find no match in the first entry, but the second entry will match. As the prefix is '-', the answer will be "no". Because a match has been made, the third entry is not processed.

Notes:

- you usually use these modifiers if most of the rights for a given user shall be specified later, but a special user or group should be treated slightly different for a few special rights.

ACLs - Default entry

There is a special ACL entry, "Default", which expands itself in-place to the default ACL.

This is useful, for example, if when you mostly want the default ACL, but with a slight modification, but you don't want to type in the default ACL all the time and you also want to be able to change the default ACL without having to edit lots of items.

Example:

```
"-NotThisGuy:write Default"
```

This will behave as usual, except that "NotThisGuy" will never be given write permission.

5.6.6 Secrets

Moin uses secrets to encrypt or cryptographically sign something like:

- tickets

Secrets are long random strings and *not* a reuse of any of your passwords. Don't use the strings shown below, they are NOT secret as they are part of the moin documentation. Make up your own secrets:

```
secrets = {
    'security/ticket': 'asdasdvarebtZertbaoihnownbrrergfqe3r',
}
```

If you don't configure these secrets, moin will detect this and reuse Flask's SECRET_KEY for all secrets it needs.

5.6.7 Groups

Group names can be used in place of usernames within ACLs. There are three types of groups: WikiGroups, ConfigGroups, and CompositeGroups. A group is a list of unicode names, where a name may be either a username or another group name.

Use of groups will reduce the administrative effort required to maintain ACL rules, especially in wikis with a large community of users. Rather than change multiple ACL rules to reflect a new or departing member, a group may be updated. To achieve maximum benefit, some advance planning is required to determine the kind and names of groups suitable for your wiki.

The wiki server must be restarted to reflect updates made to ConfigGroups and CompositeGroups.

Names of WikiGroup items must end in "Group". There is no such requirement for the names of ConfigGroups or CompositeGroups.

Group backend configuration

The WikiGroups backend is enabled by default so there is no need to add the following to wikiconfig:

```
def groups(self):
    from moin.datastructures import WikiGroups
    return WikiGroups()
```

To create a WikiGroup that can be used in an ACL rule:

- Create a wiki item with a name ending in "Group" (the content of the item is not relevant)
- Edit the metadata and add an entry for "usergroup" under the heading "Extra Metadata (JSON)":

```
{
  "itemid": "36b6cd973d7e4daa9cfa265dcf751e79",
  "namespace": "",
  "usergroup": [
    "JaneDoe",
    "JohnDoe"
  ]
}
```

- Use the new group name in one or more ACL rules.

The ConfigGroups backend uses groups defined in the configuration file. Adding the following to wikiconfig creates an EditorGroup and an AdminGroup and prevents the use of any WikiGroups:

```
def groups(self):
    from moin.datastructures import ConfigGroups
    groups = {'EditorGroup': ['AdminGroup', 'John', 'JoeDoe', 'Editor1'],
              'AdminGroup': ['Admin1', 'Admin2', 'John']}
    return ConfigGroups(groups)
```

CompositeGroups enable both ConfigGroups and WikiGroups to be used. The example below defines the same ConfigGroups used above and enables the use of WikiGroups. Note that order matters! Since ConfigGroups backend is first in the return tuple, the EditGroup and AdminGroup defined below will be used should there be WikiGroup items with the same names:

```
def groups(self):
    from moin.datastructures import ConfigGroups, WikiGroups, CompositeGroups
    groups = {'EditorGroup': ['AdminGroup', 'John', 'JoeDoe', 'Editor1'],
              'AdminGroup': ['Admin1', 'Admin2', 'John']}
    return CompositeGroups(ConfigGroups(groups), WikiGroups())
```

Dict backend configuration

The dict backend provides a means for translating phrases in documentation through the use of the GetVal macro.

The WikiDicts backend is enabled by default so there is no need to add the following to wikiconfig:

```
def dicts(self):
    from moin.datastructures import WikiDicts
    return WikiDicts()
```

To create a WikiDict that can be used in an GetVal macro:

- Create a wiki item with a name ending in "Dict" (the content of the item is not relevant)
- Edit the metadata and add an entry for "somedict" under the heading "Extra Metadata (JSON)":

```
{
  "itemid": "332458ceab334991868de8970980494e",
  "namespace": "",
  "somedict": {
    "apple": "red",
    "banana": "yellow",
    "pear": "green"
  }
}
```

The ConfigDicts backend uses dicts defined in the configuration file. Adding the following to wikiconfig creates a OneDict and a NumbersDict and prevents the use of any WikiDicts:

```
def dicts(self):
    from moin.datastructures import ConfigDicts
    dicts = {'OneDict': {'first_key': 'first item',
                         'second_key': 'second item'},
             'NumbersDict': {'1': 'One',
                             '2': 'Two'}}
    return ConfigDicts(dicts)
```

CompositeDicts enable both ConfigDicts and WikiDicts to be used. The example below defines the same ConfigDicts used above and enables the use of WikiDicts. Note that order matters! Since ConfigDicts backend is first in the return tuple, the OneDict and NumbersDict defined below will be used should there be WikiDict items with the same names:

```
def dicts(self):
    from moin import ConfigDicts, WikiDicts, CompositeDicts
    dicts = {'OneDict': {'first_key': 'first item',
                         'second_key': 'second item'},
             'NumbersDict': {'1': 'One',
                             '2': 'Two'}}
    return CompositeDicts(ConfigDicts(dicts),
                          WikiDicts())
```

5.6.8 Storage

MoinMoin supports storage backends as different ways of storing wiki items.

Setup of storage is rather complex and layered, involving:

- Routing middleware that dispatches by namespace to the respective backend
- ACL checking middleware that makes sure nobody accesses something he/she is not authorized to access
- Indexing mixin that indexes some data automatically on commit, so items can be selected / retrieved faster.
- storage backends that store wiki items

create_simple_mapping

This is a helper function to make storage setup easier when your wiki will be using only the predefined namespaces and one kind of backend (OS file system, sqla or sqlite). It creates a simple setup that defines storage backends for these namespaces:

- default - items that define wiki content
- users - personnal user content
- userprofiles - user metadata such as timezone, subscriptions, etc.
- help-en - English language help pages for editors
- help-common - media items used by help-en

For each namespace, the following structures are created:

- configure ACLs protecting the namespaces
- setup router middleware that dispatches to the namespace backends
- setup a indexing mixin that maintains an index of all namespaces

Call it as follows:

```
from moin.storage import create_simple_mapping

namespace_mapping, backend_mapping, acl_mapping = create_simple_mapping(
    uri=...,
    default_acl=dict(before=...,
                     default=...,
                     after=...,
                     hierarchic=..., ),
    users_acl=dict(before=...,
                   default=...,
                   after=...,
```

(continues on next page)

(continued from previous page)

```
                    hierarchiv=False, ),
    userprofiles_acl=dict(before=...,
                          default=...,
                          after=...,
                          hierarchiv=False, ),
)
```

The *_acl variables are dictionaries specifying the ACLs for each namespace. See the docs about ACLs.

The *uri* depends on the kind of storage backend and stores you want to use, see below. Usually it is a URL-like string in the form of:

```
stores:fs:/srv/mywiki/%(backend)s/%(kind)s
```

stores is the name of the backend, followed by a colon, followed by a store specification. *fs* is the type of the store, followed by a specification that makes sense for the fs (filesystem) store, i.e. a path with placeholders.

%(backend)s placeholder will be replaced by the namespace for the respective backend. *%(kind)s* will be replaced by 'meta' or 'data' later.

The mapping created will look like this:

Namespace	Filesystem path for storage
default	/srv/mywiki/default/
users	/srv/mywiki/users/
userprofiles	/srv/mywiki/userprofiles/
help-en	/srv/mywiki/help-en/
help-common	/srv/mywiki/help-common/

If your wiki will be using custom namespaces then you cannot use the *create_simple_mapping* method. See the *create_mapping* method in the *namespaces* section below.

protecting middleware

Features:

- protects access to lower storage layers by ACLs (Access Control Lists)
- makes sure there won't be ACL security issues, even if upper layers have bugs
- if you use create_simple_mapping, you just give the ACL parameters; the middleware will be set up automatically by moin.

routing middleware

Features:

- dispatches storage access to different backends depending on the namespace
- if you use create_simple_mapping, the router middleware will be set up automatically by moin.

indexing middleware

Features:

- maintains an index for important metadata values
- speeds up looking up / selecting items
- makes it possible for lower storage layers to be simpler
- the indexing middleware will be set up automatically by moin.

stores backend

This is a backend that ties together 2 stores to form a backend: one for meta, one for data

fs store

Features:

- stores into the filesystem
- store metadata and data into separate files/directories

Configuration:

```
from moin.storage import create_simple_mapping

data_dir = '/srv/mywiki/data'
namespace_mapping, acl_mapping = create_simple_mapping(
    uri='stores:fs:{0}/%(nsname)s/%(kind)s'.format(data_dir),
    default_acl=dict(before='WikiAdmin:read,write,create,destroy',
                     default='All:read,write,create',
                     after='', ),
    users_acl=dict(before='WikiAdmin:read,write,create,destroy',
                   default='All:read,write,create',
                   after='', ),
    # userprofiles is for internal use, contains only user metadata, access denied to␣
↪all
    userprofiles_acl=dict(before='All:',
                          default='',
                          after='', ),
)
```

sqla store

Features:

- stores data into an (SQL) database / table
- can either use 1 database per store or 1 table per store and you need to give different table names then
- uses slqalchemy (without the ORM) for database abstraction
- supports multiple types of databases, for example:
  - sqlite (default, comes built-into Python)
  - postgresql
  - mysql
  - and others, see sqlalchemy docs.

uri for *create_simple_mapping* looks like e.g.:

```
stores:sqla:sqlite:////srv/mywiki/data/mywiki_%(nsname)s_%(kind).db
stores:sqla:sqlite:////srv/mywiki/data/mywiki_%(nsname)s.db::%(kind)s
stores:sqla:mysql://myuser:mypassword@localhost/mywiki_%(nsname)s::%(kind)s
stores:sqla:postgres://myuser:mypassword@localhost/mywiki_%(nsname)s::%(kind)s
```

The uri part after "sqla:" is like:

```
DBURI::TABLENAME
```

Please see the sqlalchemy docs about the DBURI part.

Grant 'myuser' (his password: 'mypassword') full access to these databases.

sqlite store

Features:

- directly talks to sqlite, without using sqlalchemy
- stores data into an sqlite database, which is a single file
- can either use 1 database per store or 1 table per store and you need to give different table names then
- can optionally compress/decompress the data using zlib: default compression level is 0, which means "do not compress"

uri for *create_simple_mapping* looks like e.g.:

```
stores:sqlite:/srv/mywiki/data/mywiki_%(nsname)s_%(kind)s.db
stores:sqlite:/srv/mywiki/data/mywiki_%(nsname)s.db::%(kind)s
stores:sqlite:/srv/mywiki/data/mywiki_%(nsname)s.db::%(kind)s::1
```

The uri part after "sqlite:" is like:

```
PATH::TABLENAME::COMPRESSION
```

It uses "::" as separator to support windows pathes which may have ":" after the drive letter.

memory store

Features:

- keeps everything in RAM
- if your system or the moin process crashes, all data is lost, so definitely not for production use
- mostly intended for testing
- single process only

fileserver backend

Features:

- exposes a part of the filesystem as read-only wiki items
  - files will show up as wiki items

        * with 1 revision
        * with as much metadata as can be made up from the filesystem metadata
    - directories will show up as index items, listing links to their contents
- might be useful together with SMBMount pseudo-authenticator

namespaces

Moin has support for multiple namespaces. You can configure them per your needs. URLs for items within a namespace are similar to sub-items.

To configure custom namespaces, start by adding these imports near the top of wikiconfi.py:

```
from moin.storage import create_mapping
from moin.constants.namespaces import NAMESPACE_DEFAULT, NAMESPACE_USERPROFILES,␣
↪NAMESPACE_USERS
```

Next, find the section in wikiconfig.py that looks similar to this:

```
namespace_mapping, backend_mapping, acl_mapping = create_simple_mapping(
    uri='stores:fs:{0}/%(backend)s/%(kind)s'.format(data_dir),  # orig
    default_acl=dict(before='',
                     default='All:read,write,create,destroy,admin',
                     after='',
                     hierarchic=False, ),
    users_acl=dict(before='',
                   default='All:read,write,create,destroy,admin',
                   after='',
                   hierarchic=False, ),
    # userprofiles contain only metadata, no content will be created
    userprofiles_acl=dict(before='All:',
                          default='',
                          after='',
                          hierarchic=False, ),
)
```

and replace all of the above with this:

```
uri = 'stores:fs:{0}/%(backend)s/%(kind)s'.format(data_dir),  # use file system for␣
↪storage
# uri='stores:sqlite:{0}/mywiki_%(backend)s_%(kind)s.db'.format(data_dir),  # sqlite,␣
↪1 table per db
# uri='stores:sqlite:{0}/mywiki_%(backend)s.db::%(kind)s'.format(data_dir),  # sqlite,
↪ 2 tables per db
# sqlite via SQLAlchemy
# uri='stores:sqla:sqlite:///{0}/mywiki_%(backend)s_%(kind)s.db'.format(data_dir),  #␣
↪ 1 table per db
# uri='stores:sqla:sqlite:///{0}/mywiki_%(backend)s.db:%(kind)s'.format(data_dir),  #␣
↪2 tables per db

namespaces = {
    # maps namespace name -> backend name
    # these 3 standard namespaces are required
    NAMESPACE_DEFAULT: 'default',
    NAMESPACE_USERS: 'users',
    NAMESPACE_USERPROFILES: 'userprofiles',
```

(continues on next page)

(continued from previous page)

```
    # trailing / below causes foo, and bar to be stored in default backend
    'foo/': 'default',
    'bar/': 'default',
    # custom namespace with a backend - note absence of trailing /
    'baz': 'baz',
}
backends = {
    # maps backend name -> storage
    # feature to use different storage types for each namespace is not implemented so␣
↪use None below.
    # the storage type for all backends is set in 'uri' above,
    # all values in `namespace` dict must be defined as keys in `backends` dict
    'default': None,
    'users': None,
    'userprofiles': None,
    # required for baz namespace defined above
    'baz': None,
}
acls = {
    # maps namespace name -> acl configuration dict for that namespace
    NAMESPACE_USERPROFILES: dict(before='All:',
                                 default='',
                                 after='',
                                 hierarchic=False, ),
    NAMESPACE_USERS: dict(before='SuperUser:read,write,create,destroy,admin',
                          default='All:read,write,create',
                          after='',
                          hierarchic=False, ),
    NAMESPACE_DEFAULT: dict(before='SuperUser:read,write,create,destroy,admin',
                          default='All:read,write,create',
                          after='',
                          hierarchic=False, ),
    'foo': dict(before='SuperUser:read,write,create,destroy,admin',
                default='All:read,write,create',
                after='',
                hierarchic=False, ),
    'bar': dict(before='SuperUser:read,write,create,destroy,admin',
                default='All:read,write,create',
                after='',
                hierarchic=False, ),
    'baz': dict(before='SuperUser:read,write,create,destroy,admin',
                default='All:read,write,create',
                after='',
                hierarchic=False, ),
}
namespace_mapping, backend_mapping, acl_mapping = create_mapping(uri, namespaces,␣
↪backends, acls, )

# define mapping of namespaces to item_roots (home pages within namespaces).
root_mapping = {'foo': 'fooHome'}
# default root, use this value in case a particular namespace key is not present in␣
↪the above mapping.
default_root = 'Home'
```

Edit the above renaming or deleting the lines with foo, bar, and baz and adding the desired custom namespaces. Be sure all the names in the *namespaces* dict are also added to the *acls* dict. All of the values in the namespaces dict must be included as keys in the backends dict.

There cannot be an item with the same name as a namespace. Using the example above, if import19 is used to convert a moin 1.9 wiki to moin 2.0, then an item *foo* would be renamed to *foo/fooHome*.

5.6.9 Mail configuration

Sending E-Mail

Moin can optionally send E-Mail. Possible uses:

- send out item change notifications
- enable users to reset forgotten passwords
- inform admins about runtime exceptions

You need to configure some settings before sending E-Mail can be supported:

```
# the "from:" address [Unicode]
mail_from = "wiki <wiki@example.org>"

# a) using an SMTP server, e.g. "mail.provider.com" with optional `:port`
appendix, which defaults to 25 (set None to disable mail)
mail_smarthost = "smtp.example.org"

# if you need to use SMTP AUTH at your mail_smarthost:
#mail_username = "smtp_username"
#mail_password = "smtp_password"

# b) alternatively to using SMTP, you can use the sendmail commandline tool:
#mail_sendmail = "/usr/sbin/sendmail -t -i"
```

Todo: describe more moin configuration

Admin Traceback E-Mails

If you want to enable admins to receive Python tracebacks, you need to configure the following:

```
# list of admin emails
admin_emails = ["admin <admin@example.org>"]

# send tracebacks to admins
email_tracebacks = True
```

Please also check the logging configuration example in *docs/examples/config/logging/email*.

User E-Mail Address Verification

At account creation time, Moin can require new users to verify their E-Mail address by clicking a link that is sent to them.

Make sure that Moin is able to send E-Mails (see previous section) and add the following line to your configuration file to enable this feature:

```
user_email_verification = True
```

5.7 Framework Configuration

Things you may want to configure for Flask and its extensions (see their docs for details):

```
# for Flask
SECRET_KEY = 'you need to change this so it is really secret'
DEBUG = False  # use True for development only, not for public sites!
TESTING = False  # if true, some servers will detect file changes and restart
#SESSION_COOKIE_NAME = 'session'
#PERMANENT_SESSION_LIFETIME = timedelta(days=31)
#USE_X_SENDFILE = False
#LOGGER_NAME = 'MoinMoin'

# for Flask-Caching:
#CACHE_TYPE = 'filesystem'
#CACHE_DIR = '/path/to/flask-cache-dir'
#CACHE_THRESHOLD = 300  # expiration time in seconds
```

5.8 Logging Configuration

By default, logging is configured to emit output on *stderr*. This will work well for the built-in server (it will show up on the console) or for Apache2 and similar (logging will be put into error.log).

Logging is very configurable and flexible due to the use of the *logging* module of the Python standard library.

The configuration file format is described there:

http://www.python.org/doc/current/library/logging.html#configuring-logging

There are also some logging configurations in the *docs/examples/config/logging/* directory.

Logging configuration needs to be done very early, usually it will be done from your adaptor script, e.g. moin.wsgi:

```
from moin import log
log.load_config('wiki/config/logging/logfile')
```

You have to fix that path to use a logging configuration matching your needs (use an absolute path).

Please note that the logging configuration has to be a separate file, so don't try this in your wiki configuration file!

5.9 Changes in MoinMoin

5.10 MoinMoin Version History

Please note: Starting from the MoinMoin version you used previously, you should read all more recent entries (or at least everything marked with HINT).

Version 2.0.0alpha: Fixes: * ...

New features: * ...

Other changes: * ...

Todo: rewrite CHANGES in rst syntax

5.11 Upgrading

Note: Internally, moin2 is very different than moin 1.x.

moin 2.0 is *not* just a +0.1 step from 1.9 (like 1.8 -> 1.9), but the change of the major version number is indicating *major and incompatible changes*.

So please consider it to be different and incompatible software that tries to be compatible in some areas:

- Server and wiki engine Configuration: expect to review/rewrite it
- Wiki content: expect 90% compatibility for existing moin 1.9 content.
  - The most commonly used simple moin wiki markup (like headlines, lists, bold) has not changed
  - CamelCase auto links will be converted to explicit [[CamelCase]] links
  - [[attachment:my.jpg]] will be converted to [[/my.jpg]]
  - {{attachment:my.jpg}} will be converted to {{/my.jpg}}
  - expect to change custom macros, parsers, action links, 3rd party extensions

5.11.1 From moin < 1.9

If you run an older moin version than 1.9, please first upgrade to a recent moin 1.9.x version (preferably >= 1.9.7) before upgrading to moin2. You may want to run that for a while to be sure everything is working as expected.

Note: Both moin 1.9.x and moin2 are WSGI applications. Upgrading to 1.9 first also makes sense concerning the WSGI / server side.

5.11.2 From moin 1.9.x

If you want to keep your user's password hashes and migrate them to moin2, make sure you use moin >= 1.9.7 WITH enabled passlib support and that all password hashes stored in user profiles are {PASSLIB} hashes. Other hashes will get removed in the migration process and users will need to do password recovery via email (or with admin help, if that does not work).

Backup

Have a backup of everything, so you can go back in case it doesn't do what you expect. If you have a testing machine, it is a good idea to try it there first and not directly modify your production machine.

Install moin2

Install and configure moin2, make it work, and start configuring it from the moin2 sample config. Do *not* just use your 1.9 wikiconfig.

Adjusting the moin2 configuration

It is essential that you edit wikiconfig.py before you import your 1.9 data. In particular, review the settings for:

```
- sitename
- interwikiname
- SECRET_KEY
- secrets
- default_acl
- users_acl
```

Clean up your moin 1.9 data

It is a good idea to clean up your 1.9 data first, before trying to import it into moin2. In doing so you can avoid quite some warnings that the moin2 importer would produce.

You do this with moin *1.9*, using these commands:

```
moin ... maint cleanpage
moin ... maint cleancache
```

Deleted pages will not be migrated. A message will be written to the log for each deleted page.

Importing your moin 1.9 data

Assuming you have no moin2 storage and no index directories created yet, include the -s and -i options to create the storage and an index.

The import19 argument to the *moin* script will read your 1.9 data_dir (pages, attachments and users), convert the data, write it to your moin2 storage and build the index:

```
moin import19 -s -i --data_dir /<path to moin1.9>/wiki/data 1>import19out.log 2>
↪import19err.log
```

If you use the command as given, it will write all stdout and stderr output into two log files. Please review them to find out whether the importer had critical issues with your data.

By default, all items using moin 1.9 markup are converted to moin 2 markup. The converted revision will have a timestamp one second later than the last revision's timestamp to preserve revision history.

Page revisions that were created with leading *#format creole* and *#format rst* commands will retain the creole and rst markups.

There is an additional option to convert pages with moin wiki markup using one of the other moin2 output converters: markdown, rst, html, or docbook. Add the *–markup_out* or *-m* option to the *moin import19* command above. To convert the last revision of all pages with moin wiki markup to markdown:

```
-m markdown
```

The import19 process will create a wiki directory structure different from moin 1.9. There will be three namespaces under /wiki/data: "default", "userprofiles", and "users". Each namespace will have "data" and "meta" subdirectories. Additional custom namespaces can be created by editing wikiconfig.py.

Most of the data from the 1.9 pages directory will be converted to the "default" directory. User home pages and subpages will be converted to the "users" directory. The data from the 1.9 "users" directory will be converted to the "userprofiles" directory. The "userprofiles" directory contains data used internally and should always be protected from any access by ACLs.

Testing

Review the logs for error messages. Start the moin server and try the "Index" and "History" views to see what is included. Check whether your data is complete and rendering correctly.

Links to user's home items and subitems will be broken after the migration and must be corrected manually. Links similar to *[[JoeDoe]]* and *[[JoeDoe/SubItem]]* should be corrected to *[[users/JoeDoe]]* and *[[users/JoeDoe/SubItem]]* respectively.

If you find issues with data migration from moin 1.9 to 2, please check the moin2 issue tracker.

Keep your backups

Make sure you keep all backups of your moin 1.9 installation, such as code, config, data, just in case you are not happy with moin2 and need to revert to the old version.

Converting after reverting

The import19 process converts text items using Moinmoin 1.9 syntax to Moinmoin 2.0 syntax.

The conversion is accomplished by creating a new revision of each moin wiki text item. Click the History link under the Item Views panel to view the revisions. The latest revision will have a content type of "Moinmoin" while the older revisions created prior to conversion will have a content type of "Moinmoin 1.9" Click the Diff link to see the content changes made by import19.

If a moin wiki item is reverted to a revision having a content type of "Moinmoin 1.9" with embedded old style CamelCase auto links and/or attachments (*{{attachment:my.jpg}}*), the revision is not converted to the Moinmoin 2 syntax automatically. Editors must do the conversion by clicking the Convert link within the Item Views panel.

Reverted revisions left in the Moinmoin 1.9 format will render correctly and the reverted item may be updated and saved using the old 1.9 syntax. However, it is recommended that all such revisions be converted to the new moin syntax because the old CamelCase and attachment conventions are deprecated and will never be included in the moin 2 docs.

5.12 Backup and Restore

5.12.1 Full Backup / Restore

The best way to recover from data loss is to have a **full** backup of your machine. With this backup you can easily restore your machine to a working condition.

The procedure below explains how to selectively backup only the files essential to your MoinMoin installation. While there is no need to maintain both a full and a selective backup, having at least one of the two is strongly recommended.

5.12.2 Selective Backup

If you want a backup of MoinMoin and your data, then backup the following:

- your data
- moin configuration, e.g. wikiconfig.py
- logging configuration, e.g. logging.conf
- moin deployment script, e.g. moin.wsgi

- web server configuration, e.g. apache virtualhost config
- optional: moin code + dependencies; you should at least know which version you ran, so you can reinstall that version when you need to restore

To create a dump of all data stored in moinmoin (wiki items, user profiles), run the following command:

```
moin save --all-backends --file backup.moin
```

Please note that this file contains sensitive data like user profiles, wiki contents, so store your backups in a safe place that no unauthorized individual can access.

5.12.3 Selective Restore

To restore all software and configuration files to their original place, create an empty wiki first:

```
moin index-create -s -i  # -s = create new storage
                         # -i = create new index
```

To load the backup file into your empty wiki, run:

```
moin load --file backup.moin
```

Then build an index of the loaded data:

```
moin index-build
```

5.13 Indexes

5.13.1 General

MoinMoin relies strongly on indexes that accelerate access to item metadata and data, and makes it possible to have simple backends, because the index layer is doing all the hard and complex work.

Indexes are used internally for many operations like item lookup, history, iterating over items, search, interactive search, etc.

MoinMoin won't be able to start with damaged, inaccessible or non-existing indexes. As a result, you will need to configure and initialize indexing correctly first.

moin will automatically update the index when items are created, updated, deleted, destroyed, or renamed via the storage api of moin, indexing layer or above.

5.13.2 Configuration

Your need to have a `index_storage` entry in your wiki config.

We use whoosh for indexing and as whoosh supports multiple storage backends, this entry is made to potentially support any storage supported by whoosh.

In general, this entry has the form of:

```
index_storage = kind, (p1, p2, ...), {kw1=..., kw2=..., ...}
```

Currently, we only support the 'FileStorage' kind of index storage, which only has one parameter - the index directory:

```
index_storage = 'FileStorage', ("/path/to/moin-2.0/wiki/index", ), {}
```

Notes for FileStorage:

- The path MUST be absolute, writable and should be on a fast, local filesystem.
- Moin will use *index.temp* directory as well, if you build an index at the *temporary location*.

5.13.3 moin index script reference

You can use the `moin index-*` group of script commands to manage indexes.

Many of the script commands for index management support a *–tmp* option to use the temporary index location. This is useful if you want to do index operations in parallel to a running wiki which is still using the index at the normal index location.

moin index-create

Creates an empty but valid index.

Note: the moin WSGI application needs an index to successfully start up. As the moin index-* script commands are also based on the moin WSGI application, this can lead to a chicken and egg problem. To solve this, the moin command has a `-i` (`--index-create`) option that will trigger index creation on startup.

Additionally, if the storage is also non-existent yet, one might also need `-s` (`--storage-create`) to create an empty storage on startup.

moin index-build

Process all revisions of the wiki and add the indexable documents to the index.

Note:

- For big wikis, this can take rather long; consider using –tmp.
- index-build does NOT clear the index at the beginning.
- index-build does not check the current contents of the index. Therefore you must not run index-build multiple times for the same data or the same wiki.

moin index-update

Compare an index to the current storage contents and update the index as needed (add, remove, update) to reflect the current storage contents.

Note: You can use this after building at the tmp location to get the changes that happened to the wiki while building the index as well. You can run index-update multiple times to keep even more caught up.

moin index-destroy

Destroy an index, such that nothing left at the respective location.

moin index-move

Move the index from the temporary location to the normal location.

moin index-optimize

Optimize an index:: see Whoosh docs for more details.

moin index-dump

Output index contents in human readable form, e.g. for debugging purposes.

Note: only fields with attribute `stored=True` can be displayed.

5.13.4 Building an index for a single wiki

If your wiki is fresh and empty

Use:

```
moin index-create --storage-create --index-create
moin index-create -s -i  # same, but shorter
```

Storage and index are now initialized and both empty.

If you add data to your wiki, the index will get updated automatically.

If your wiki has data and is shut down

If index needs a rebuild for some reason, e.g. index lost, index damaged, incompatible upgrade, etc., use:

```
moin index-create -i
moin index-build  # can take a while...
```

If your wiki has data and should stay online

Use:

```
moin index-create -i --tmp
moin index-build --tmp  # can take a while...
moin index-update --tmp  # should be quicker, make sure we have 99.x%
# better shut down the wiki now or at least make sure it is not changed
moin index-update --tmp  # make sure we have indexed all content, should be even␣
↪quicker.
moin index-move  # instantaneously
# start the wiki again or allow changes now again
```

Note: Indexing puts load onto your server, so if you like to do regular index rebuilds, schedule them at some time when your server is not too busy.

5.13.5 Building an index for a wiki farm

If you run a wiki farm (multiple related wikis), you may share the index between the wikis, so users will be able to search in one wiki and also see results from the other wikis.

Before you start, you must prepare your wiki configs. For example, for a company that uses two farm wikis, such as `Sales` and `Engineering`, Their respective wiki configs could look like:

`Sales`:

```
interwikiname = "Sales"
index_storage = 'FileStorage', ("/path/to/moin-2.0/wiki/index", ), {}
```

`Engineering`:

```
interwikiname = "Engineering"
index_storage = 'FileStorage', ("/path/to/moin-2.0/wiki/index", ), {}
```

Now do the initial index building:

```
moin index-create -i  # create an empty index
# now add the indexes from both other wikis:
moin index-build  # with Sales wiki configuration
moin index-build  # with Engineering wiki configuration
```

Now you should have a shared index for all wikis.

Note: Do not build indexes for multiple wikis in parallel. This is not supported.

5.14 Password Resetting/Invalidation

There might be circumstances when the wiki admin wants or needs to reset one user's or all users' password (hash).

For example:

- you had a security breach on your wiki server (or somewhere else) and the old password hashes (or passwords) were exposed
- you want to make sure some user or all users set a new password, e.g. if:
  - your password policy has changed (requiring longer passwords for example)
  - you changed your passlib configuration and want to immediately have all hashes upgraded

Note: if we say "reset a password" (to use a commonly used term), we mean to "invalidate the password hash" (so that no password exists that validates against that hash). MoinMoin does not keep user passwords in cleartext.

The files we refer to below are located in docs/examples/password-reset/...

5.14.1 Resetting one or few password(s)

If you somehow interact with the users corresponding to the user accounts in question (by phone or directly), you don't need the extensive procedure as described below, just use:

```
moin account-password --name JoeDoe
```

That will reset JoeDoe's password. Tell him to visit the login URL and use the "forgot my password" functionality to define a new password.

If that doesn't work (e.g. if e-mail is not enabled for your wiki or he has a non-working e-mail address in his profile), you can also set a password for him:

```
moin account-password --name JoeDoe --password uIkV9.-a3
```

Choose a rather complicated password to make sure they change it a minute afterwards (to another, hopefully safe password).

5.14.2 Resetting many or all password(s)

If you have a lot of passwords to reset, you need a better procedure that avoids having to deal with too many users individually.

Preparing your users

Tell your users beforehands that you will be doing a password reset, otherwise they might find the automatically generated E-Mail they'll get suspicious and you'll have to explain it to them individually that the E-Mail is legitimate.

Also, remind your users that having a valid E-Mail address in their user settings is essential for getting a password recovery E-Mail.

If an active user does somehow not get such a mail, you likely will have to manually define a valid E-Mail address (or even password) for that user.

Make sure E-Mail functionality works

If you know you have working E-Mail functionality, skip this section.

Password recovery and password reset notification work via E-Mail, so you should have it configured:

```
# the E-Mail address used for From: (consider using an address that
# can be directly replied to, at least while doing the pw reset):
mail_from = 'wiki@example.org'
# your smtp mail server hostname:port (default is 25)
mail_smarthost = 'mail.example.org:587'
# the login there, if authentication is needed
mail_username = 'wiki@example.org'
mail_password = 'SuperSecretSMTPPassword'
```

You can try whether it works by using the "forgot my password" functionality on the login page.

Editing mailtemplate.txt

If you edit mailtemplate.txt, please be very careful and follow these rules (otherwise you might just see the script command crashing):

The contents must be utf-8 (or ascii, which is a subset of utf-8). In case of doubt, just use plain English.

Some places you likely should edit are marked with XXX.

Do not use any % character in your text (except for the placeholders). If you need a verbatim % character, you need to write %%.

It is a very good idea to give some URL (e.g. of a web or wiki page) in the text where users can read more information.

Of course the information at that URL should be readable without requiring a wiki login (you just have invalidated his/her password!), so the user can get informed before clicking links he got from someone via E-Mail.

We have added a wikitemplate.txt you can use to create such a wiki page.

Instead of creating a web or wiki page with the information, you could also write all the stuff into the mail template directly, but please consider that E-Mail delivery to some users might fail for misc. reasons, so having some information on the web/wiki is usually better.

Editing wikitemplate.txt

Just copy & paste it to some public page in your wiki, e.g. "PasswordReset".

Some places you likely should edit are marked with XXX.

Doing the password reset

Maybe first try it with a single user account:

```
moin account-password --name JoeDoe --notify --subject 'Wiki password reset' --text-
↪from-file mailtemplate.txt
```

Use some valid name, maybe a testing account of yourself. You should now have mail. If that worked ok, you can now do a global password reset for your wiki:

```
moin account-password --verbose --all-users --notify --subject 'Wiki password reset' -
↪-text-from-file mailtemplate.txt
```

The subject may contain a placeholder for the sitename, which is useful for wiki farms (showing the builtin default here):

```
'[%(sitename)s] Your wiki account data'
```

5.15 Moin Command Line Interface

Moin2 has two command line interfaces. The newer interface, powered by quickinstall.py and started by the **./m** command (**m** on windows), implements the most common functions used by desktop users and developers. This CLI is only available when moin is installed using git to clone a repository from https://github.com/moinwiki/moin or alternative.

The older interface, **moin**, is implemented by several Python scripts located in the /scripts/ directory. This interface targets wiki migration, account creation and maintenance, and wiki maintenance.

There is some overlap between the two interfaces. Several of the commands within the newer interface are implemented by wrapping one or more of the older interface commands to accomplish a task.

5.15.1 ./m Interface

The virtual environment must be activated before using the ./m interface. Executing **./m** (**m** on windows) without any options produces the menu:

```
usage: "{0} <target>" where <target> is:

quickinstall    update virtual environment with required packages
extras          install packages required for docs and moin development
docs            create moin html documentation (requires extras)
interwiki       refresh intermap.txt
log <target>    view detailed log generated by <target>, omit to see list

new-wiki        create empty wiki
sample          create wiki and load sample data
restore *       create wiki and restore wiki/backup.moin *option, specify file
import19 <dir> <args> import a moin1.9 wiki/data instance from <dir> with <args>
                where <args> = "--markup_out moinwiki" or markdown,rst,html,...

run *           run built-in wiki server *options (--port 8081)
backup *        roll 3 prior backups and create new backup *option, specify file
dump-html *     create a static HTML image of wiki *options, see docs
index           delete and rebuild indexes

css             run lessc to update basic theme CSS files
tests *         run tests, log output (-v -k my_test)
coding-std      correct scripts that taint the repository with trailing spaces..

del-all         same as running the 4 del-* commands below
del-orig        delete all files matching *.orig
del-pyc         delete all files matching *.pyc
del-rej         delete all files matching *.rej
del-wiki        create a backup, then delete all wiki data
```

5.15.2 moin Interface

moin is the command line interface to miscellaneous MoinMoin Wiki related tools.

If you invoke **moin** without any arguments, it will show a short quick help,

moin –help will show a more complete overview:

```
usage: moin [-c CONFIG] [-i] [-s] [-?]
            {help,moin,run,create-instance,index-create,index-build,index-update,
↪index-destroy,index-move,index-optimize
,index-dump,save,load,load-sample,dump-html,account-create,account-disable,account-
↪password,maint-reduce-revisions,maint
-set-meta,item-get,item-put,load-help,dump-help,import19,shell,runserver}
            ...

positional arguments:
  {help,moin,run,create-instance,index-create,index-build,index-update,index-destroy,
↪index-move,index-optimize,index-dum
p,save,load,load-sample,dump-html,account-create,account-disable,account-password,
↪maint-reduce-revisions,maint-set-meta,
item-get,item-put,load-help,dump-help,import19,shell,runserver}
    help
    moin                Runs the Flask development server i.e. app.run()
    run                 Runs the Flask development server i.e. app.run()
    create-instance
    index-create
```

(continues on next page)

(continued from previous page)

```
    index-build
    index-update
    index-destroy
    index-move
    index-optimize
    index-dump
    save
    load
    load-sample
    dump-html
    account-create
    account-disable
    account-password
    maint-reduce-revisions
    maint-set-meta
    item-get
    item-put
    load-help           Load an entire help namespace from distribution source.
    dump-help           Save an entire help namespace to the distribution source.
    import19
    shell               Runs a Python shell inside Flask application context. :param␣
↪banner: banner appearing at top
                        of shell when started :param make_context: a callable␣
↪returning a dict of variables used in
                        the shell namespace. By default returns a dict consisting of␣
↪just the app. :param use_ipython:
                        use IPython shell if available, ignore if not. The IPython␣
↪shell can be turned off in command
                        line by passing the **--no-ipython** flag.
    runserver           Runs the Flask development server i.e. app.run()

options:
  -c CONFIG, --config CONFIG
  -i, --index-create
  -s, --storage-create
  -?, --help            show this help message and exit
```

5.15.3 See also

moinmoin(1)

CHAPTER 6

Getting Support for and Contributing to MoinMoin

6.1 MoinMoin Supports You

6.1.1 Free Support

You can get free support and information here:

- on our chat channels, see https://moinmo.in/MoinMoinChat
- on our wiki, see https://moinmo.in/ - please note that quite a lot of content there is about moin 1.x and does not apply to moin2. One page has a lot of information about moin2 and also links to all sorts of moin2 resources: https://moinmo.in/MoinMoin2.0
- on our mailing list, see https://moinmo.in/MoinMoinMailingLists
- on github: https://github.com/moinwiki/moin

Note: All free support is done voluntarily by helpful MoinMoin community members. Thanks to everyone who is helping!

If you enjoyed / want to enjoy free community support, please also consider being an active part of the community and also supporting it.

6.1.2 Commercial Support

As MoinMoin 2.0 is not released yet, there is no support for production systems based on it.

If you want to talk about development topics, please contact the developers.

6.2 You Support MoinMoin

6.2.1 Like to help others?

Just stay connected to IRC, our wiki, the mailing list (see above) and help others searching for support there.

6.2.2 Found a bug?

- File a bug report on the issue tracker.
- Even better: fix the bug, submit a pull request with a unit test and a fix.

6.2.3 Have an idea?

- Discuss it on IRC and file a feature request.
- Even better: discuss and write some Python code implementing it.

6.2.4 Born to code?

- Help to work on moin2 core, so it gets released sooner.
- Help to maintain moin 1.9 until moin2 is ready.

6.2.5 Loving UI / UX design?

- Help us make moin2 look and feel better!

6.2.6 Have good language or documentation skills?

- If you are a native speaker of a language other than English, with a good understanding of English, consider helping with improving translation to your language. **(not yet for moin2, too early!)** see also *Translating MoinMoin*
- Improve the documentation (see below). Here is a list of all TODOs in this documentation:

Todo: rewrite CHANGES in rst syntax

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/moin-20/checkouts/latest/docs/admin/changes.rst, line 10.)

Todo: add the usual coding(s) for some platforms (like windows)

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/moin-20/checkouts/latest/docs/admin/configure.rst, line 564.)

Todo: check if SMBMount still works as documented

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/moin-20/checkouts/latest/docs/admin/configure.rst, line 709.)

Todo: describe more moin configuration

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/moin-20/checkouts/latest/docs/admin/configure.rst, line 1530.)

6.3 Translating MoinMoin

6.3.1 If your language already exists

To find out if someone has already started a translation of moin2 into your language; check the folder moin/translations in the source tree. If there is a folder with your language code (locale)[1], you can start with the steps below. If not, please take a look at *If your language doesn't exist yet*.

1. Make sure you have the latest version of the source tree (git). You will also need to have python installed, with setuptools and babel packages.
2. Go to the top directory and execute:

   ```
   python setup.py update_catalog -l <locale>
   ```

   where locale is the short language descriptor of your desired language. It should be the name of a folder in MoinMoin/translations. For German it is "de".
3. Open the file `src/moin/translations/<locale>/LC_MESSAGES/messages.po` and do your translation. A short explanation of this process follows:
   - Find an entry with an empty or bad translated text, the text after msgstr, and apply your changes.
   - **never** edit the 'msgid' string, and only edit the 'msgstr' field
   - Variables like `%(name)x`, where x is any character, must be kept as they are. They must occur in the translated text.
   - For better readability you can divide a text-string over more than one line, by "surrounding" each line with double quotes ("). It is a usual convention to have a maximal line-length of 80 characters.
   - Comments starting with "#.", "*#:*" or "*#*|" are auto-generated and should not be modified.
   - Comments starting with "# " (# and at least one whitespace) are translator-comments. You can modify/add them. They have to be placed right before the auto-generated comments.
   - Comments starting with "*#,*" and separated with "," are flags. They can be auto-generated, but they can also be set by the translator.

     An important flag is "fuzzy". It shows that the msgstr string might not be a correct translation. Only the translator can judge if the translation requires further modification, or is acceptable as it is. Once satisfied with the translation, he/she then removes this fuzzy attribute.
4. Save the messages.po file and execute:

   ```
   python setup.py compile_catalog -l <locale>
   ```

[1] For more information on locale strings, see http://www.gnu.org/software/hello/manual/gettext/Locale-Names.html.

Guidelines for translators

In languages where a separate polite form of address exists, like the German "Sie"/"Du", always use the polite form.

6.3.2 If your language doesn't exist yet

You want to translate moin2 to your language? Great! Get in contact with the developers, but ...

Note: please don't ask us whether we want other translations, we currently do not want them, it is still too early. We just want 1 translation and it needs to be German because that is what many moin developers can maintain themselves.

---

1. Initialize a new catalog:

```
python setup.py init_catalog -l <locale>
```

2. Adjust the `src/moin/translations/<locale>/LC_MESSAGES/messages.po`.

   Follow the instructions in *First steps with a new *.po file* and then you can remove the fuzzy flag, which prevents the file from being compiled.

3. Follow the steps above, see *If your language already exists*.

First steps with a new `*.po` file

A newly created translation needs a few initial preparations:

- replace "`PROJECT`" with "`MoinMoin 2`"
- replace "`FIRST AUTHOR <EMAIL@ADDRESS>`" with the appropriate information about yourself
- replace "`PROJECT VERSION`" in the head msgstr with "`MoinMoin 2.0`" or newer if neccessary
- change the value of "`Last-Translator`" to your data
- change the value of "`Language-Team`" to "`Language <moin-user@lists.sourceforge.net>`"

6.3.3 Note for developers

Since we support newstyle gettext there is no need to use the `format()`-Method in internationalized Strings anymore. An example will explain this: instead of `_('Hello %(name)s!') % dict(name='World')` you can just write `_('Hello %(name)s!', name='World')`.

If the translatable string contains a variable plural, that means the string contains an object which you don't know the exact quantity of, then you will have to use `ngettext()`. Note that this is not only needed for the decision between one and more objects, because other languages have other and more difficult plurals than English. The usage is `ngettext(singular, plural, num, **variables)`. `**variables` enables you to use the newstyle form as explained above.

For example: `ngettext("%(number)d file removed from %(directory)s", "%(number)d files removed from %(directory)s", num=n, number=n, directory=directory)`

`n` has to appear twice because the first gives ngettext() information about the exact number and the second is the variable for the format string replacement.

If you made changes to any gettext() string, please update the .pot file using:

```
python setup.py extract_messages
```

Because this sometimes creates large diffs, just because of a change in line numbers, you can of course use this command sparingly. Another option for better readability is to do a separate commit for this.

CHAPTER 7

Developing of MoinMoin

7.1 Development

7.1.1 Useful Resources

IRC channels on chat.freenode.net (quick communication and discussion):

- #moin-dev (core development topics)
- #moin (user support, extensions)

Wikis:

- https://moinmo.in/ (production wiki, using moin 1.9)

Documentation (installation, configuration, user docs, api reference):

- https://readthedocs.org/docs/moin-20/en/latest/

Repository, Issue tracker (bugs, proposals, todo), Code Review, etc.:

- https://github.com/moinwiki/moin

7.1.2 Requirements for development

The *virtualenv* Python package is required. The installation process for *virtualenv* varies with your OS and Python distribution. Many linux distributions have a package manager that may do the installation. Windows users (and perhaps others) may download setuptools from https://pypi.org/project/setuptools/. Once setuptools is installed, do "*easy_install virtualenv*". Current ActiveState distributions include virtualenv in the installation bundle. If all else fails, try your favorite search engine.

git is required should you wish to contribute patches to the moin2 development effort. Even if you do not intend to contribute, git is highly recommended as it will make it easy for you to obtain fixes and enhancements from the moin2 repositories. git can be installed with most linux package managers or downloaded from https://git-scm.com/. You can also find GUI clients there.

7.1.3 Typical development workflow

This is the typical workflow for anyone that wants to contribute to the development of Moin2.

create your development environment

- if you do not have a github account, create one at https://github.com/
- fork the main repository: https://github.com/moinwiki/moin to your gh user
- clone your gh repo to your local development machine:

```
cd <parent_directory_of_your_future_repo>
git clone https://github.com/yourname/moin.git
```

- cd to repo root:

```
cd moin
```

- create the virtualenv and download packages:

```
python quickinstall.py
```

- activate virtualenv:

```
. activate  # Windows: activate
```

- create a wiki instance and load sample data:

```
./m sample  # Windows: m sample
```

- start the built-in server:

```
./m run  # Windows: m run
```

- point your browser at http://127.0.0.1:8080/ to access your development wiki
- key ctrl+C to stop the built-in server

add more tools, exercise tools

- install additional software that developers may require:

```
./m extras  # Windows: m extras
```

- run the unit tests, note any existing test failures:

```
./m tests  # Windows: m tests
```

- install NodeJS and NPM with Linux package manager; Windows users may download both from https://nodejs.org/download/
  - On Ubuntu 14.04 or any distribution based on Ubuntu you need to install “npm” and “nodejs-legacy” (to get the “node” command).
- install lessc (“less” below is not a typo):

```
sudo npm install less -g  # Windows: npm install less -g
lessc --version"  # show version number to prove it works
```

- regenerate CSS files:

```
./m css  # Windows: m css
git diff  # verify nothing changed
```

- check for coding errors (tabs, trailing spaces, line endings, template indentation and spacing):

```
./m coding-std  # Windows: m coding-std
git diff  # verify nothing changed
```

- revert any changes from above:

```
git reset --hard
```

- create local docs:

```
./m docs  # Windows: m docs
```

- set options on your favorite editor or IDE
  - convert tabs to 4 spaces
  - delete trailing blanks on file save
  - use unix line endings (use Windows line endings on .bat and .cmd files)
  - use mono-spaced font for editing
- if you are new to git, read about it (https://git-scm.com/book/), consider printing a cheatsheet
- if you want a Python IDE, try https://www.jetbrains.com/pycharm/ Free Community Edition
- join #moin-dev IRC channel; ask questions, learn what other developers are doing

review configuration options

- review https://moin-20.readthedocs.org/en/latest/admin/configure.html
- following the instructions in wikiconfig.py, create wikiconfig_local.py and wikiconfig_editme.py
- configure options by editing wikiconfig_editme.py
  - set superuser privileges on at least one username
  - the default configuration options are commonly used, it is likely new bugs can be found by testing different options

find a task to work on

- look at the issue tracker to find a task you can solve
- in case you find a new bug or want to work on some (non-trivial) new issue or idea that is not on the issue tracker, create an issue with a detailed description
- discuss your chosen task with other developers on the #moin-dev IRC channel
- to avoid duplicate work, add a comment on the issue tracker that you are working on that issue

- just before you start to code changes, bring your repo up to date:

```
git checkout master        # make sure you are on master branch
git pull mm master         # update your master branch
git checkout -b mychange   # create a new branch "mychange"
...                        # implement your change
tox                        # run the tests, fix any new failure!
git status                 # check what new files you created
git diff                   # check what changes you did
git add ...                # add the files you want to commit
git commit                 # commit, write a nice commit comment
git push                   # push to your gh user's moin repo
...                        # go to gh moinwiki/moin and make a PR
```

develop a testing strategy

- if you fix something that had no test, first try to write a correct, but failing test for it, then fix the code and see a successful test
- if you implement new functionality, write tests for it first, then implement it
- make a plan for using a browser to test your changes; which wiki pages are effected, how many browsers must be tested

develop a working solution

- work in your local repo on your local development machine (be sure you work in the right branch)
- concentrate on one issue / one topic, create a clean set of changes (that means not doing more than needed to fix the issue, but also it means fixing the issue completely and everywhere)
- write good, clean, easy-to-understand code
- obey PEP-8
- do not fix or change code unrelated to your task, if you find unrelated bugs, create new issues on the tracker
- regularly run the unit tests ("./m tests"), the amount of failing tests shall not increase due to your changes

review your working solution

- use git diff, git status - read everything you changed - slowly, look for things that can be improved
  - if you have TortoiseGIT, use those graphical tools to review changes
- look for poor variable names, spelling errors in comments, accidental addition or deletion of blank lines, complex code without comments, missing/extra spaces
- fix everything you find before requesting feedback from others
- run tests again "./m tests"
- check for trailing spaces, line endings, template indentation "./m coding-std"
- if Javascript files were changed, run https://jslint.com/

publish your change

- do some final testing - practically and using the unit tests
- commit your changes to your local repo, use a concise commit comment describing the change
  - while a commit message may have multiple lines, many tools show only 80 characters of the first line
  - stuff as much info as possible into those first 80 characters:

```
<concise description of your change>, fixes #123
```

- push the changeset to your public github repo
- create a pull request so your changes will get reviewed and pulled into the main repository
- if you fixed an issue from the issue tracker, be sure the issue gets closed after your fix has been pulled into main repo.
- celebrate, loop back to "find a task to work on"

update your virtualenv

Every week or so, do "m quickinstall" to install new releases of dependent packages. If any new packages are installed, do a quick check for breakages by running tests, starting the build-in server, modify an item, etc.

7.1.4 MoinMoin architecture

moin2 is a WSGI application and uses:

- flask as framework
  - flask-script for command line scripts
  - flask-babel / babel / pytz for i18n/l10n
  - flask-theme for theme switching
  - flask-caching as cache storage abstraction
- werkzeug for low level web/http page serving, debugging, builtin server, etc.
- jinja2 for templating, such as the theme and user interface
- flatland for form data processing
- EmeraldTree for xml and tree processing
- blinker for signalling
- pygments for syntax highlighting
- for stores: filesystem, sqlite3, sqlalchemy, memory
- jquery javascript lib, a simple jQuery i18n plugin Plugin
- CKeditor, the GUI editor for (x)html
- TWikiDraw, AnyWikiDraw, svgdraw drawing tools

7.1.5 How MoinMoin works

This is a very high level overview about how moin works. If you would like to acquire a more in-depth understanding, please read the other docs and code.

WSGI application creation

First, the moin Flask application is created; see *moin.app.create_app*:

- load the configuration (app.cfg)
- register some modules that handle different parts of the functionality
  - moin.apps.frontend - most of what a normal user uses
  - moin.apps.admin - for admins
  - moin.apps.feed - feeds, e.g. atom
  - moin.apps.serve - serving some configurable static third party code
- register before/after request handlers
- initialize the cache (app.cache)
- initialize index and storage (app.storage)
- initialize the translation system
- initialize theme support

This app is then given to a WSGI compatible server somehow and will be called by the server for each request for it.

Request processing

Let's look at how it shows a wiki item:

- the Flask app receives a GET request for /WikiItem
- Flask's routing rules determine that this request should be served by *moin.apps.frontend.show_item*.
- Flask calls the before request handler of this module, which:
  - sets up the user as flaskg.user - an anonymous user or logged in user
  - initializes dicts/groups as flaskg.dicts, flaskg.groups
  - initializes jinja2 environment - templating
- Flask then calls the handler function *moin.apps.frontend.show_item*, which:
  - creates an in-memory Item
    - by fetching the item of name "WikiItem" from storage
    - it looks at the contenttype of this item, which is stored in the metadata
    - it creates an appropriately typed Item instance, depending on the contenttype
  - calls Item._render_data() to determine what the rendered item looks like as HTML
  - renders the *show_item.html* template and returns the rendered item html
  - returns the result to Flask
- Flask calls the after request handler which does some cleanup

- Flask returns an appropriate response to the server

Storage

Moin supports different stores, like storing directly into files / directories, using key/value stores, using an SQL database etc, see *moin.storage.stores*. A store is extremely simple: store a value for a key and retrieve the value using the key + iteration over keys.

A backend is one layer above. It deals with objects that have metadata and data, see *moin.storage.backends*.

Above that, there is miscellaneous functionality in *moin.storage.middleware* for:

- routing by namespace to some specific backend
- indexing metadata and data + comfortable and fast index-based access, selection and search
- protecting items by ACLs (Access Control Lists)

DOM based transformations

How does moin know what the HTML rendering of an item looks like?

Each Item has some contenttype that is stored in the metadata, also called the input contenttype. We also know what we want as output, also called the output contenttype.

Moin uses converters to transform the input data into the output data in multiple steps. It also has a registry that knows all converters and their supported input and output mimetypes / contenttypes.

For example, if the contenttype is *text/x-moin-wiki;charset=utf-8*, it will find that the input converter handling this is the one defined in *converters.moinwiki_in*. It then feeds the data of this item into this converter. The converter parses this input and creates an in-memory *dom tree* representation from it.

This dom tree is then transformed through multiple dom-to-dom converters for example:

- link processing
- include processing
- smileys
- macros

Finally, the dom-tree will reach the output converter, which will transform it into the desired output format, such as *text/html*.

This is just one example of a supported transformation. There are quite a few converters in *moin.converters* supporting different input formats, dom-dom transformations and output formats.

Templates and Themes

Moin uses jinja2 as its templating engine and Flask-Themes as a flask extension to support multiple themes. There is a `moin/templates` directory that contains a base set of templates designed for the Modernized theme. Other themes may override or add to the base templates with a directory named `themes/<theme_name>/templates`.

When rendering a template, the template is expanded within an environment of values it can use. In addition to this general environment, parameters can also be given directly to the render call.

Each theme has a `static/css` directory. Stylesheets for the Basic theme in MoinMoin are compiled using the source `theme.less` file in the Basic theme's `static/custom-less` directory.

```
./m css  # Windows: m css
```

Internationalization in MoinMoin's JS

Any string which has to be translated and used in the JavaScript code, has to be defined at `moin/templates/dictionary.js`. This dictionary is loaded when the page loads and the translation for any string can be received by passing it as a parameter to the _ function, also defined in the same file.

For example, if we add the following to `i18n_dict` in `dictionary.js`

```
"Delete this"  : "{{  _("Delete this") }}",
```

The translated version of "somestring" can be accessed in the JavaScript code by

```
var a = _("Delete this");
```

7.1.6 Testing

We use pytest for automated testing. It is currently automatically installed into your virtualenv as a dependency.

Running the tests

To run all the tests, the easiest way is to do:

```
./m tests  # windows:  m tests
```

To run selected tests, activate your virtual env and invoke pytest from the toplevel directory:

```
pytest --pep8  # run all tests, including pep8 checks
pytest -rs  # run all tests and output information about skipped tests
pytest -k somekeyword  # run the tests matching somekeyword only
pytest --pep8 -k pep8  # runs pep8 checks only
pytest sometests.py  # run the tests contained in sometests.py
```

Tests output

Most output is quite self-explanatory. The characters mean:

```
. test ran OK
s test was skipped
E error happened while running the test
F test failed
x test was expected to fail (xfail)
```

If something goes wrong, you will also see tracebacks in stdout/stderr.

Writing tests

Writing tests with *pytest* is easy and has little overhead. Just use the *assert* statements.

For more information, please read: https://docs.pytest.org/

7.1.7 Documentation

Sphinx (https://www.sphinx-doc.org) and reST markup are used for documenting moin. Documentation reST source code, example files and some other text files are located in the *docs/* directory in the source tree.

Creating docs

Sphinx can create all kinds of documentation formats. The most common are the local HTML docs that are linked to under the User tab. To generate local docs:

```
./m docs  # Windows: m docs
```

7.1.8 Moin Shell

While the make.py utility provides a menu of the most frequently used commands, there may be an occasional need to access the moin shell directly:

```
source <path-to-venv>/bin/activate  # or ". activate"  windows: "activate"
moin -h                             # show help
```

CHAPTER 8

Autogenerated API docs

CHAPTER 9

Indices and Tables

- genindex
- modindex
- search
- glossary