MoinMoin Documentation

Release 2.0.0a0

The MoinMoin developers

Apr 22, 2017

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

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

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.

Who is using MoinMoin?

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

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.

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

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.

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)

Authentication

- Builtin - username / password login form of moin, MoinAuth
- Builtin HTTP Basic Auth - browser login form, HTTPAuthMoin

- OpenID - relying party, OpenIDAuth
- Auth against LDAP / Active Directory (LDAPAuth)
- Any authentication your web server supports via GivenAuth

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

Anti-Spam

- TextChas (text captchas)
- Form Ticketing

Storage

Item Types

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

Storage Backend Types

- file system
- sql database, such as sqlite3 or everything supported by SQLalchemy
- Kyoto Tycoon / Kyoto Cabinet
- mongodb
- you can easily add your own backend with little code

Serialization

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

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

User Interface

OO user interface

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

Templating

- Theme support / User interface implemented with templates

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

Markup support

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

Feeds

- Atom
- Google Sitemap

Notification

- by email: smtp or sendmail

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

Logging

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

Technologies

- html5, css, javascript with jquery, svg
- python
- flask, flask-caching, flask-babel, flask-themes, 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
 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
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
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
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
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.

    <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

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.

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.

OpenID This is an optional field where you may enter an OpenID to be associated with your account. OpenID provides a common mechanism for websites to authenticate users and store data about them like username and real name. If you have an OpenID, you may want to enter it here.

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.

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, alias and OpenID.

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.

OpenID If you have an OpenID which you would like to associate with your account, enter it here.

Timezone Setting this value will allow you to see edit times as they would appear in your 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.

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.

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.

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`. his will show a list of modifications you have made to wiki items.

MOINTODO +mychanges only links to the item which you edit, not the specific revision. If you edit and item several times, it just inserts several identical links to that item. This behaviour should be checked and rectified.

MOINTODO +mychanges isn't very pretty if you visit it without making any changes, it just says "My Changes" at the top with the rest of the page left blank.

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.

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.

Markups Supported by MoinMoin

Moin Wiki markup overview

The report follows the moin 1.9 help page and reports syntaxes that do not match 1.9 help syntax documentation. The structure and order has been matched with other markup rst files namely creoleWiki.rst and mediaWiki.rst at http://hg.moinmo.in/moin/2.0-dev/file/42d8cde592fb/docs/user

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

Internal Links

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

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

Markup	Comment
`{{example.png}}`	Embed example.png inline
`{{https://static.moinmo.in/logos/moinmoin.png}}`	Embed example.png inline
`{{ItemName}}`	Transclude (embed the contents of) ItemName inline.
`{{/SubItem}}`	Transclude SubItem inline.
`{{ 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, PIL needs to be installed.
`{{ https://moinmo.in/ || width=800 }}`	Resizes the `object` which is embedded using HTML tags. Also markup involving '&' parameters like `&w` doesn't make much sense.

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 **RSTTODO**

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 chosen in rst

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

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).

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||
```

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**

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

- 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
- Comments: comment

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"
`<<Include(ItemOne/SubItem)>>`	Embeds the contents of `ItemOne/SubItem` inline
`<<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.
`<<PageNameList()>>`	Inserts names of all wiki items
`<<RandomItem(3)>>`	Inserts names of 3 random items
`<<TableOfContents(2)>>`	Shows a table of contents up to level 2
`<<Verbatim(`same` __text__)>>`	Inserts text as entered

Smileys and Icons

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

This is wiki markup in a **div** with css *class="red solid"*. |

Notes:

- The div cannot be shown in reStructuredText, so a table cell has been made to demonstrate the border produced. In MoinMoin, this border will appear red.

Admonitions

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:

> **Warning: 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.

Comments

Markup:

```
{{{#!wiki comment/dotted
This is a wiki parser section with class "comment dotted" (see HelpOnParsers).

Its visibility gets toggled the same way.
}}}
```

Result:

This is a wiki parser section with class "comment dotted" (see HelpOnParsers).
Its visibility gets toggled the same way.

Notes:

- reStructuredText has no support for dotted borders, so a table cell is used to illustrate the border which will be produced. This markup will actually produce a dotted border in MoinMoin.
- The toggle display feature does not work yet

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 **RSTTODO**.

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

RSTTODO: 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
.

RSTTODO: 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!')

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"
`<<Include(ItemOne/SubItem)>>`	Embeds the contents of `ItemOne/SubItem` inline
`<<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.
`<<PageNameList()>>`	Inserts names of all wiki items
`<<RandomItem(3)>>`	Inserts names of 3 random items
`<<TableOfContents(2)>>`	Shows a table of contents up to level 2
`<<Verbatim(`same` __text__)>>`	Inserts text as entered

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:

```
To reference another item with this wiki, do http:Home or `Home <http:Home>`_. (If␣
↪this page
is not a wiki page, then the Home links are not valid.)

To reference an anchor within this item, first create an invisible anchor:

.. _example:

Then reference the anchor, like this: example_

To link to section headings do Headings_. If a section title has embeded blanks,␣
↪you must
enclose the heading with backtic characters: `Internal Links`_.
```

Result:

To reference another item with this wiki, do http:Home or Home. (If this page is not a wiki page, then the Home links are not valid.)

To reference an anchor within this item, first create an invisible anchor: Then reference the anchor, like this: *example*

To link to section headings do *Headings*. If a section title has embeded blanks, you must enclose the heading with backtic characters: *Internal Links*.

Notes

- Section titles 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.

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

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 2.1.1

- item 3
```

Result:

- item 1
- item 2
  - item 2.1
  - item 2.1.1
- item 3

Ordered Lists

Markup:

```
1. item 1

   (A) item 1.1
   (#) item 1.2

       i) item 1.2.1
       #) item 1.2.2

#. item 2
```

Result:

1. item 1
   (a) item 1.1
   (b) item 1.2

    i. item 1.2.1

    ii. item 1.2.2

2. item 2

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.
- 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:

<table>
<tr><td>A
B</td><td rowspan="2">D</td></tr>
<tr><td colspan="2">C</td></tr>
</table>

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)

Note 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:

Literals 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
```

Docbook XML Markup

Warning: **MOINTODO** fill this page with useful content (see other markup related pages about how it should look like)

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 **RSTTODO**.

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.

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'''`

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

Hyperlinks

Internal links

RSTTODO These link targets are not interpreted. (The examples shown here result in empty links) **RSTTODO** 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.

RSTTODO 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.

RSTTODO 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
`—`	—
`←`	←
`→`	→
`⇐`	
`⇒`	
`©`	©

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

RSTTODO colspan is not interpreted

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 to create or edit issues on the moin issue tracker.

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

This page, describing the Markdown syntax, is written in RST. Instances where RST cannot duplicate the same rendering produced by Markdown are flagged with **RST NOTE**. The RST 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
--- 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

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`.

RST NOTE:

RST has no means of forcing a line break, so the above example showing a line ending with 2 spaces fails. RST does not support bold-italics so the above example fails.

Linking

Markup:

```
A simple way to write a link: [MoinMoin](http://moinmo.in) or [PNG](png)
Another way to write a link is to use a reference: [favorite comic][xkcd]

References can be defined anywhere in the document, but it will not be visible in␣
↪the rendered HTML:
[xkcd]: http://xkcd.com

Naked URLs like http://moinmo.in are not supported. URLs and email addresses may␣
↪be enclosed in angle brackets: <http://moinmo.in> and <me@example.com>.
```

Result:

A simple way to write a link: MoinMoin or PNG Another way to write a link is to use a reference: favorite comic

References can be defined anywhere in the document, but it will not be visible in the rendered HTML:

Naked URLs like *http://moinmo.in* are not supported. URLs and email addresses may be enclosed in angle brackets: http://moinmo.in and me@example.com.

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
```

Result:

- apples
- oranges
- **pears**
  - carrot
  - **beet**
    - man
    - woman

RST NOTE: While Sphinx renders the above correctly, the Moin RST parser shows pears and beet in bold text.

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
```

Result:

1. apples
2. oranges
3. pears
   (a) carrot
   (b) beet
      i. man
      ii. woman

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.

RST NOTE: The Moin RST 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.

RST NOTE: Moin's RST parser markup confuses nested blockquotes rendering with term: definition syntax. Sphinx renders it correctly.

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](png "Optional title")

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

Transcluding a page from a remote site works, but there is no way to change the␣
↪browser's default size:

![Alt text](http://test.moinmo.in/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:

Transcluding a page from a remote site works, but there is no way to change the browser's default size:

RST NOTE: The Moin RST parser renders all four images above. The Sphinx parser renders only the external png image from http://static.moinmo.in/logos/moinmoin.png. RST syntax does 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: This feature 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:

E = MC²

This word is bold.

This word is italic.

This word is bold.

This word is bold; colors depend upon configuration settings.

RST NOTE: The RST parser does not allow inline HTML markup so none of the above examples are rendered as they would be in Moin’s Markdown.

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 three is      |right-aligned  |$1600 |Necklace|
|col 2 is          |centered       |$12   |Gloves  |
|col 4 is          |left-aligned   |$100  |Hat     |

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

Result:

RST NOTE: RST tables are broken on the Moin RST parser. Although table rendering on Sphinx is correct, there is no means of specifying cell alignment; therefore, none of the examples are shown.

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);
```
```

Result:

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

MDTODO: Syntax highlighting is not working. [#169](https://bitbucket.org/thomaswaldmann/moin-2.0/issue/169/highlight-python-does-not-work) documents a similar issue for the moinwiki converter.

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

Result:

A class of green (that will create a green background per a CSS rule) is added to this paragraph.

RST NOTE: The background of the above paragraph is not green because there is no option in RST syntax to assign a class.

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.

RST NOTE: Two line definition lists do not render as two line definitions in RST.

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 the document. Its markup language can also be changed at any time by modifying the item's `contenttype` metadata. Currently Moin2 supports MoinWiki, WikiCreole, reStructuredText, Docbook, MediaWiki and Markdown markups.

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

MOINTODO: Currently the use of `{{{#!syntax content}}}` parsers crashes moin. This should be looked into.

MOINTODO: The creation of items/editing of an item's metadata is not yet documented. This is beyond the scope of this index page and should be looked into.

Searching and Finding

Entering search queries

Usually there is a simple but rather short search query input field offered by the theme. By submiting a query it will search in item names and content, but only in the current contents, not in non-current revisions, and display the search results to you.

On the search results view you will get a bigger search query input field, for example for refining your query, and you may also choose to additionally search in non-current item revisions. Selecting that will search in all revisions.

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

or (does the same):

wiki AND moin
```

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

```
wiki OR moin
```

Search for documents containing "wiki" and NOT "bad":

```
wiki NOT bad
```

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*
```

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"
```

Searching in specific fields

As long as you do not specify otherwise, moin will search in `name`, `name_exact` and `content` fields.

To specify the field to search in, just use the *fieldname:searchterm* syntax.

Field name	Field value
`wikiname`	wiki name, e.g. ITWiki, EngineeringWiki, SalesWiki
`name`	document name, e.g. Home, MyWikiPage
`name_exact`	same as `name`, but is not tokenized
`content`	document contents, e.g. This is some example content.
`contenttype`	document type, e.g. text/plain;charset=utf-8
`tags`	tags of the document, e.g. important, hard, todo
`language`	(main) language of the document contents, e.g. en
`mtime`	document modification (submission) time, 201112312359
`username`	submitter user name, e.g. JoeDoe
`address`	submitter IP address, e.g. 127.0.0.1
`hostname`	submitter DNS name, e.g. foo.example.org
`acl`	access control list (see below)
`itemlinks`	link targets of the document, e.g. OtherItem
`itemtransclusions`	transclusion targets of the document, e.g. OtherItem

Examples

Search in metadata fields:

```
contenttype:text
contenttype:image/jpeg
tags:todo
mtime:201108
address:127.0.0.1
language:en
hostname:localhost
```

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
```

Notes

moin uses indexed search. Keep in mind that this has some special properties:

- By using an index, the search is usually rather 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:

- "foobar" is put into the index somehow
- you search for "ooba" - you will not find it, because only "foobar" was put into the index
- solution: search for "foobar": fast and will find it
- solution: search for "*ooba*" or r".*ooba.*": slow, but will find it

More information

See the Whoosh query language docs.

Namespaces

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`

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*.

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::.**

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

Requirements

MoinMoin requires Python 2.7.x. 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 2.7 installed by default or will have a package manager that will install Python 2.7 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/.

The *virtualenv* Python package is also 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.python.org/pypi/setuptools. Once setuptools is installed, do "*easy_install virtualenv*". Current ActiveState distributions include virtualenv in the installation bundle. If all else fails, try Google.

Mercurial (hg) is required should you wish to contribute patches to the moin2 development effort. Even if you do not intend to contribute, Mercurial is highly recommended as it will make it easy for you to obtain fixes and enhancements from the moin2 repositories. Mercurial can be installed with most linux package managers or downloaded from http://mercurial.selenic.com/. As an alternative, most Windows users will prefer to install TortoiseHG (includes Mercurial) from http://tortoisehg.bitbucket.org/.

Servers

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

- the builtin "./m run" or "moin" server 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 "./m run -d -r" to turn off the werkzeug debugger and auto reloader. See Werkzeug docs for more information.

Dependencies

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

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.

Downloading and Installing

Downloading

The recommended way to download moin2 is to clone the moin2 Mercurial repository or its mirror. Open a terminal window or a command prompt, cd to the directory that will hold your project root directory and enter either one of the commands below:

```
hg clone http://hg.moinmo.in/moin/2.0 moin-2.0

OR

hg clone http://bitbucket.org/thomaswaldmann/moin-2.0 moin-2.0
```

Now make sure your work directory is using the default branch:

```
hg up -C default
```

An alternative installation method is to download the bz2 archive from http://hg.moinmo.in/moin/2.0 and unpack it. Once unpacked, continue to follow the instructions below.

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 2.7 executable. This is usually just "python", but may be "python2.7", "/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.

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
docs            create moin html documentation
extras          install OpenID, Pillow, pymongo, sqlalchemy, ldap, upload.py
interwiki       refresh contrib/interwiki/intermap.txt (hg 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
import <dir>    import a moin 1.9 wiki/data instance from <dir>
index           delete and rebuild indexes

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 *option, specify directory

css             run Stylus and lessc to update theme CSS files
tests *         run tests, output to pytest.txt *options (-v -k my_test)
coding-std      correct scripts that taint the repository with trailing spaces..
api             update moin api docs (files are under hg version control)
dist            delete wiki data, then create distribution archive in dist/

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
```

While most of the above menu choices may be executed now, new users should do:

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

to create a wiki instance and load it with sample data. Next, run the built-in wiki server:

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

As the server starts, about 20 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.

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* (or *wikiconfig_editme.py*) and change the settings for:

- interwikiname
- SECRET_KEY
- secrets

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

- *m docs* - to create docs, see User tab, Documentation (local)
- *m extras* - to install Pillow for manipulating images
- *m del-wiki* - get rid of the sample data
- *m new-wiki* or *m import ...* - no data or moin 1.9 data
- *m backup* - backup wiki data as needed or as scheduled

Warning: Backing up data at this point may provide a false sense of security because no migration tool has been developed to migrate data between moin2 versions. In its current alpha state, there may be code changes that impact the structure of the wiki data or indexes. Should this occur, you must start over with an empty wiki and somehow copy and paste the contents of all the old wiki items into the new wiki. While no such changes are planned, they have happened in the past and may happen in the future.

If you installed moin2 by cloning the Moin2 Mercurial repository, then you will likely want to install updates on a periodic basis. To determine if there are updates available, open a terminal window or command prompt, cd to your project root, and enter the command below:

```
hg incoming
```

If there are any updates, a brief description of each update will be displayed. To add the updates to your cloned repository, do:

```
hg pull -u
```

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 run         # in Unix
```

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.

ActiveState Python

While ActiveState bundles pip and virtualenv in its distribution, there are two missing files. The result is the following error messages followed by a traceback:

```
Cannot find sdist setuptools-*.tar.gz
Cannot find sdist pip-*.tar.gz
```

To install the missing files, do the following and then rerun "python quickinstall.py":

```
\Python27\Scripts\pip.exe uninstall virtualenv
\Python27\Scripts\easy_install virtualenv
```

Other Issues

If you encounter some other issue not described above, try researching the unresolved issues at https://bitbucket.org/thomaswaldmann/moin-2.0/issues?status=new&status=open.

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.

Server Options

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):
./m run  # Windows:  "m run"

# or, if you need another configuration file, ip address, or port:
./m run --config /path/to/wikiconfig.py --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.:

```
./m run --host 0.0.0.0 --port 80 -d -r
```

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.

Introduction into MoinMoin Configuration

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.

Directory Structure

Shown below are parts of the directory structure after cloning moin or unpacking a release. 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-2.0/                    # clone root or unpack directory
    contrib/
        interwiki/
            intermap.txt     # interwiki map: created by cloning or unpacking,␣
↪updated by "./m interwiki"
    docs/
        _build/
            html/            # local copy of moin documentation, created by␣
↪running "./m docs" command
    MoinMoin/                # large directory containing moin application code
    wiki/                    # the wiki instance; created by running "./m sample"␣
↪or "./m new-wiki" commands
        data/                # wiki data is stored here
        index/               # wiki indexes are stored here
    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
```

wikiconfig.py Layout

A wikiconfig.py looks like this:

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

class Config(DefaultConfig):
    # some comment
    sometext = u'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 u'your value' whereby the *u'...'* means that this is a unicode string.
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.

Wiki Engine Configuration

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 *MoinMoin/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() -%}
{%- 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

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
- contributors do not mind logging in to edit
- errant bots are overloading your server by following the wiki navigation links on every page.

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

- Replace the Home/moin/creole/markdown links in snippets.html with links to the key pages within your wiki.
- If an index to all wiki items is wanted, leave the index link as is, else remove.
- If a link to login is wanted, leave that section as is, else remove the entire block.
- Test by logging in and setting "cms" as your preferred theme.
- After testing, make the "cms" theme the default theme by adding `theme_default = u"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 for CMS theme - see configuration docs for tips on customizing cms theme
↪#}
{% macro cms_header() %}
    <div id="moin-header">
        {% block header %}
            <div id="moin-logo">
                <a href="{{ url_for('frontend.show_item', item_name=cfg.root_
↪mapping.get('', cfg.default_root)) }}">
                    {{ logo() }}
                </a>
            </div>
            <a class="moin-sitename" href="{{ url_for('frontend.show_item', item_
↪name=cfg.root_mapping.get('', cfg.default_root)) }}">
                {{ cfg.sitename }}
            </a>
            <br>
            <ul class="moin-header-links">

                {# wiki admins will want to replace these links with key item␣
↪names present in local wiki #}
                <li><a href="{{ url_for('frontend.show_item', item_name='Home') }}
↪">Start</a></li>
                <li><a href="{{ url_for('frontend.show_item', item_name='moin') }}
↪">Moin Wiki Syntax</a></li>
                <li><a href="{{ url_for('frontend.show_item', item_name='creole') }
↪}">Creole Wiki Syntax</a></li>
                <li><a href="{{ url_for('frontend.show_item', item_name='markdown
↪') }}">Markdown Wiki Syntax</a></li>
                <li><a href="{{ url_for('frontend.show_item', item_name='+index') }
↪}">Index</a></li>

                {% if request.user_agent %} {# true if browser, false if run as ./
↪m dump-html script #}
                    {% if user.valid -%}
                        {% if user.auth_method in cfg.auth_can_logout %}
                            <li><a href="{{ url_for('frontend.show_item', item_
↪name='+logout') }}">Logout</a></li>
                        {% endif %}
                        <li><a href="{{ url_for('frontend.show_item', item_name=
↪'+usersettings') }}">Settings</a></li>
                    {% else %}
                        <li><a href="{{ url_for('frontend.show_item', item_name=
↪'+login') }}">Login</a></li>
                    {%- endif %}
                {%- endif %}
```

```
            </ul>
        {% endblock %}
    </div>
    <br>
{% 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',
    '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.

- activate the virtual environment and do *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 MoinMoin/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 MoinMoin/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 MoinMoin/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.

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 MoinMoin.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 MoinMoin.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 MoinMoin.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*.

OpenID

With OpenID moin can re-use the authentication done by some OpenID provider (like Google, Yahoo, Microsoft or others):

```
from MoinMoin.auth.openidrp import OpenIDAuth
auth = [OpenIDAuth()]
```

By default OpenID authentication accepts all OpenID providers. If you like, you can configure what providers to allow (which ones you want to trust) by adding their URLs to the trusted_providers keyword of OpenIDAuth. If left empty, moin will allow all providers:

```
# Allow google profile OpenIDs only:
auth = [OpenIDAuth(trusted_providers=['https://www.google.com/accounts/o8/ud?
↪source=profiles'])]
```

To be able to log in with OpenID, the user needs to have his OpenID stored in his user profile.

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 MoinMoin.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
    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 MoinMoin.auth import MoinAuth
from MoinMoin.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=<MoinMoin.user.
↪User at 0x90a0f0c name:u'ThomasWaldmann' valid:1> kw={'username': u
↪'ThomasWaldmann', 'openid': None, 'attended': True, 'multistage': None, 'login_
↪password': u'secret', 'login_username': u'ThomasWaldmann', 'password': u'secret',
↪ 'login_submit': u''}
2011-02-05 16:35:04,716 INFO MoinMoin.auth.log:22 session: user_obj=<MoinMoin.user.
↪User at 0x90a0f6c name:u'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:u'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 MoinMoin.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: u'/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 = u"S:"  # where //server/share is usually mounted for your␣
↪windows users (display purposes only)
```

Todo

check if SMBMount still works as documented

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.
- OpenID: please check yourself.

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.

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 MoinMoin.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", ],
)
```

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 two built in functions that are protected by ACLs: superuser and notextcha:

- superuser - used for miscellaneous administrative functions. Give this only to highly trusted people
- notextcha - if you have TextChas enabled, users with the notextcha capability won't get questions to answer. Give this to known and trusted users who regularly edit in your wiki.

Example:

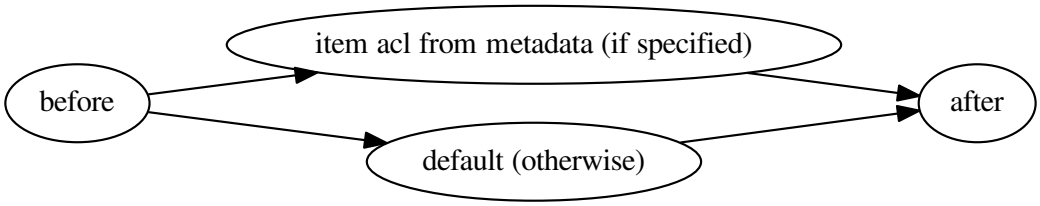
```
acl_functions = u'YourName:superuser TrustedEditorGroup:notextcha'
```

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=u'SuperUser:read,write,create,destroy,admin',
                 default=u'TrustedEditorGroup:read,write,create,destroy,admin␣
↪Known:read,write,create',
                 after=u'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.

Supported capabilities (rights):

- read - read content
- write - write (edit, modify) 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

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:

```
u"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:

```
u"+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:

```
u"-NotThisGuy:write Default"
```

This will behave as usual, except that "NotThisGuy" will never be given write permission.

Anti-Spam

TextChas

A TextCHA is a pure text alternative to ''CAPTCHAs''. MoinMoin uses it to prevent wiki spamming and it has proven to be very effective.

Features:

- when registering a user or saving an item, it can ask a random question
- moin matches the given answer against a regular expression
- questions and answers can be configured in the wiki config
- multi language support: a user gets a textcha in his language or in the language_default or in English, depending on availability of questions and answers for the language

TextCha Configuration

Tips for configuration:

- have 1 word / 1 number answers
- ask questions that normal users of your site are likely to be able to answer
- do not ask overly complex questions
- do not ask "computable" questions, like "1+1" or "2*3"
- do not ask overly obvious questions
- do not share your questions with other sites / copy questions from other sites (or spammers might try to adapt to this)
- you should at least give textchas for 'en' or for your language_default, if that is not 'en', as this will be used as fallback if MoinMoin does not find a textcha in the user's language
- if a determined bot learns the answers, create new textchas

In your wiki config, do something like this:

```
textchas = {
    'en': { # silly english example textchas (do not use them!)
            u"Enter the first 9 digits of Pi.": ur"3\.14159265",
            u"What is the opposite of 'day'?": ur"(night|nite)",
            # ...
    },
    'de': { # some german textchas
            u"Gib die ersten 9 Stellen von Pi ein.": ur"3\.14159265",
            u"Was ist das Gegenteil von 'Tag'?": ur"nacht",
            # ...
    },
    # you can add more languages if you like
}
```

Note that users with 'notextcha' ACL capability won't get TextChas to answer.

Secrets

Moin uses secrets to encrypt or cryptographically sign something like:

- textchas
- 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/textcha': 'kjenrfiefbeiaosx5ianxouanamYrnfeorf',
    '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.

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 MoinMoin.datastruct 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 MoinMoin.datastruct import ConfigGroups
    groups = {u'EditorGroup': [u'AdminGroup', u'John', u'JoeDoe', u'Editor1'],
              u'AdminGroup': [u'Admin1', u'Admin2', u'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 MoinMoin.datastruct import ConfigGroups, WikiGroups, CompositeGroups
    groups = {u'EditorGroup': [u'AdminGroup', u'John', u'JoeDoe', u'Editor1'],
              u'AdminGroup': [u'Admin1', u'Admin2', u'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 MoinMoin.datastruct 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 MoinMoin.datastruct import ConfigDicts
    dicts = {u'OneDict': {u'first_key': u'first item',
                          u'second_key': u'second item'},
             u'NumbersDict': {u'1': 'One',
                              u'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 MoinMoin.datastruct import ConfigDicts, WikiDicts, CompositeDicts
    dicts = {u'OneDict': {u'first_key': u'first item',
                          u'second_key': u'second item'},
             u'NumbersDict': {u'1': 'One',
                              u'2': 'Two'}}
    return CompositeDicts(ConfigDicts(dicts),
                          WikiDicts())
```

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. It helps you to:

- create a simple setup that uses 2 storage backends internally for these namespaces:
  - default
  - userprofiles
- configure ACLs protecting these namespaces
- setup a router middleware that dispatches to these backends
- setup a indexing mixin that maintains an index

Call it as follows:

```
from MoinMoin.storage import create_simple_mapping

namespace_mapping, backend_mapping, acl_mapping = create_simple_mapping(
    uri=...,
    default_acl=dict(before=...,
                     default=...,
                     after=...,
                     hierarchic=..., ),
    userprofiles_acl=dict(before=...,
                          default=...,
                          after=...,
                          hierarchiv=False, ),
)
```

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 'default' or 'userprofiles' for the respective backend. *%(kind)s* will be replaced by 'meta' or 'data' later.

In this case, the mapping created will look like this:

Namespace	Filesystem path for storage
default	/srv/mywiki/default/
userprofiles	/srv/mywiki/userprofiles/

default_acl and *userprofiles_acl* are dictionaries specifying the ACLs for this part of the namespace (normal content, user profiles). See the docs about ACLs.

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 MoinMoin.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=u'WikiAdmin:read,write,create,destroy',
                     default=u'All:read,write,create',
                     after=u'', ),
    userprofiles_acl=dict(before=u'WikiAdmin:read,write,create,destroy',
                          default=u'',
                          after=u'', ),
)
```

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.

kc store

Features:

- uses a Kyoto Cabinet file for storage
- very fast

- single-process only, local only

uri for *create_simple_mapping* looks like e.g.:

```
stores:kc:/srv/mywiki/data/%(nsname)s_%(kind)s.kch
```

Please see the kyoto cabinet docs about the part after *kc:*.

If you use kc with the builtin server of moin, you cannot use the reloader. Disable it with the commandline option:

```
moin moin -r
```

kt store

Features:

- uses a Kyoto Tycoon server for storage
- fast
- multi-process, local or remote.

mongodb store

Features:

- uses mongodb for storage

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 as per your need. A sample configuration looks like e.g:

```
import os

from wikiconfig import *

from MoinMoin.storage import create_mapping
from MoinMoin.constants.namespaces import NAMESPACE_DEFAULT, NAMESPACE_USERPROFILES

class LocalConfig(Config):
    wikiconfig_dir = os.path.abspath(os.path.dirname(__file__))
    instance_dir = os.path.join(wikiconfig_dir, 'wiki')
    data_dir = os.path.join(instance_dir, 'data')

    index_storage = 'FileStorage', (os.path.join(instance_dir, "index"), ), {}

    uri = 'stores:fs:{0}/%(backend)s/%(kind)s'.format(data_dir)
    namespaces = {
        # maps namespace name -> backend name
        # first, configure the required, standard namespaces:
        NAMESPACE_DEFAULT: u'default',
        NAMESPACE_USERPROFILES + '/': u'userprofiles',
        # then some additional custom namespaces:
        u'foo/': u'default',
        u'bar/': u'default',
        u'baz/': u'default',
    }
    backends = {
        # maps backend name -> storage
        u'default': None,
        u'userprofiles': None,
    }
    acls = {
        # maps namespace name -> acl configuration dict for that namespace
        NAMESPACE_USERPROFILES + '/': dict(before=u'',
                                           default=u'All:read,write,create,destroy,
↪admin',
                                           after=u'',
                                           hierarchic=False, ),
        NAMESPACE_DEFAULT: dict(before=u'',
                                default=u'All:read,write,create,destroy,admin',
                                after=u'',
                                hierarchic=False, ),
        u'foo/': dict(before=u'',
                      default=u'All:read,write,create,destroy,admin',
                      after=u'',
                      hierarchic=False, ),
        u'bar/': dict(before=u'',
                      default=u'All:read,write,create,destroy,admin',
                      after=u'',
                      hierarchic=False, ),
        u'baz/': dict(before=u'',
                      default=u'All:read,write,create,destroy,admin',
                      after=u'',
                      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 = {u'foo': u'fooHome'}
    # default root, use this value in case a particular namespace key is not␣
↪present in the above mapping.
    default_root = u'Home'
```

```
MOINCFG = LocalConfig
DEBUG = False
```

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 = u"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 = [u"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
```

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
```

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 MoinMoin 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!

Changes in MoinMoin

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

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, ...) will still work, but expect to change macros, parsers, action links, 3rd party extensions, for example.

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.

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 adjust the wiki config before you import your 1.9 data:

Example configuration:

```
from os.path import join
from MoinMoin.storage import create_simple_mapping

interwikiname = u'...' # critical, make sure it is same as in 1.9!
sitename = u'...' # same as in 1.9
item_root = u'...' # see page_front_page in 1.9

# if you had a custom passlib_crypt_context in 1.9, put it here
```

```
# configure backend and ACLs to use in future
# TODO
```

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
```

Todo

add more info about handling of deleted pages

Importing your moin 1.9 data

Assuming you have no moin2 storage and no index created yet, include the -s and -i options to create the storage and an index.

The import19 argument to the *moin* script will then read your 1.9 data_dir (pages, attachments and users), convert the data as needed, and write it to your moin2 storage and also build the index:

```
moin import19 -s -i --data_dir /your/moin/1.9/data 1>import1.log 2>import2.log
```

If you use the command as given, it will write all output into two log files. Please review them to find out whether the importer had critical issues with your data.

Testing

Start moin now, as it should have your data available.

Try “Index” and “History” views to see what is included.

Check whether your data is complete and working fine.

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.

Backup and Restore

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.

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.

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
```

Indexes

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.

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*.

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.

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.

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 = u"Sales"
index_storage = 'FileStorage', ("/path/to/moin-2.0/wiki/index", ), {}
```

`Engineering`:

```
interwikiname = u"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.

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/...

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).

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'
```

Moin Command Line Interface

Moin2 has two command line interfaces. The newer interface, powered by make.py and started by the **./m** command (**m** on windows), implements the most common functions used by desktop users and developers.

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.

./m Interface

It is not necessary to activate the virtual environment before using the ./m interface. Executing **./m** (**m** on windows) without any options produces the menu:

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

quickinstall    update virtual environment with required packages
docs            create moin html documentation
extras          install OpenID, Pillow, pymongo, sqlalchemy, ldap, upload.py
interwiki       refresh contrib/interwiki/intermap.txt (hg 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
import <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 *option, specify directory
index           delete and rebuild indexes
```

```
css             run Stylus and lessc to update theme CSS files
tests *         run tests, output to pytest.txt *options (-v -k my_test)
coding-std      correct scripts that taint the repository with trailing spaces..
api             update moin api docs (files are under hg version control)

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
```

moin Interface

moin is the command line interface to miscellaneous MoinMoin Wiki related tools. The virtual environment must be activated before running these commands:

```
. activate     # Unix
activate       # Windows
```

If you invoke **moin** without any arguments, it will start the builtin server and you'll have moin running! This is a shortcut for invoking **moin moin**.

moin --help will give a list of available subcommands.

moin <subcommand> --help will give help for some subcommand:

```
usage: moin-script.py [-c CONFIG] [-i] [-s] [-?]
    {load,index-optimize,maint-reduce-revisions,index-move,dump-html,item-get,
    index-build,account-password,index-dump,runserver,shell,index-destroy,
    account-disable,item-put,account-create,moin,index-update,save,
    index-create,maint-set-meta,import19}
                        ...

positional arguments:
    {load,index-optimize,maint-reduce-revisions,index-move,dump-html,item-get,
    index-build,account-password,index-dump,runserver,shell,index-destroy,
    account-disable,item-put,account-create,moin,index-update,save,
    index-create,maint-set-meta,import19}

    load
    index-optimize
    maint-reduce-revisions
    index-move
    dump-html
    item-get
    index-build
    account-password
    index-dump
    runserver           Runs the Flask development server i.e. app.run()
    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.
    index-destroy
    account-disable
    item-put
    account-create
```

```
    moin                Runs the Flask development server i.e. app.run()
    index-update
    save
    index-create
    maint-set-meta
    import19

optional arguments:
  -c CONFIG, --config CONFIG
  -i, --index-create
  -s, --storage-create
  -?, --help            show this help message and exit
```

See also

moinmoin(1)

CHAPTER 6

Getting Support for and Contributing to MoinMoin

MoinMoin Supports You

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 our issue tracker: https://bitbucket.org/thomaswaldmann/moin-2.0/issues
- from our repository: https://bitbucket.org/thomaswaldmann/moin-2.0/overview

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.

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.

You Support MoinMoin

Like to help others?

Just stay connected to IRC, our wiki, the mailing list (see above) and help others searching for support there.

Found a bug?

- File a bug report on the issue tracker on bitbucket.
- Even better: fix the bug, file a bug report and submit a patch and consider adding a unit test

Have an idea?

- Discuss it on IRC and file a feature request on bitbucket.
- Even better: discuss and write some Python code implementing it.

Born to code?

- Help to work on moin2 core, so it gets released sooner.
- Help to maintain moin 1.9 until moin2 is ready.

Loving UI / UX design?

- Help us make moin2 look and feel better!

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 531.)

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 695.)

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 1553.)

Todo

add more info about handling of deleted pages

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/moin-20/checkouts/latest/docs/admin/upgrade.rst, line 84.)

Translating MoinMoin

If your language already exists

To find out if someone has already started a translation of moin2 into your language; check the folder MoinMoin/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 (hg). 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 `MoinMoin/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.

[1] For more information on locale strings, see http://www.gnu.org/software/hello/manual/gettext/Locale-Names.html.

4. Save the messages.po file and execute:

```
python setup.py compile_catalog -l <locale>
```

Guidelines for translators

In languages where a separate polite form of address exists, like the German "Sie"/"Du", always use the polite form.

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 `MoinMoin/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>`"

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 `_(u'Hello %(name)s!') % dict(name='World')` you can just write `_(u'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

Development

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)
- http://test.moinmo.in/ (test wiki, using moin 2)

Documentation (installation, configuration, user docs, api reference):

- http://readthedocs.org/docs/moin-20/en/latest/

Issue tracker (bugs, proposals, todo):

- http://bitbucket.org/thomaswaldmann/moin-2.0/issues

Code Repositories (using Mercurial DVCS http://mercurial.selenic.com/):

- http://hg.moinmo.in/moin/2.0 (main repository)
- http://bitbucket.org/thomaswaldmann/moin-2.0 (bitbucket mirror for your convenience, simplifying forking and contributing)

Code review (always use this to get feedback about code changes):

- http://code.google.com/p/rietveld/wiki/CodeReviewHelp
- http://codereview.appspot.com/ (list of current codereview activity)

Pastebin (temporary storage - do not use for code review or any long-term need):

- http://rn0.ru/

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 bitbucket account, create one at https://bitbucket.org
- fork the main repository on bitbucket: https://bitbucket.org/thomaswaldmann/moin-2.0
- clone the main repository to your local development machine:

```
cd <parent_directory_of_your_future_repo>
hg clone https://bitbucket.org/thomaswaldmann/moin-2.0 moin-2.0
```

- ensure you are in default branch:

```
hg update default
```

- create the virtualenv and download packages:

```
python quickinstall.py
```

- 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, including upload.py:

```
./m extras  # Windows: m extras
```

- if you do not have a google account, create one at http://codereview.appspot.com
- read about code review at: http://code.google.com/p/rietveld/wiki/CodeReviewHelp
- practice using codereview by making a trivial change to any source file, do "python upload.py –oauth2"
  - inspect your patch set at http://codereview.appspot.com
  - experiment by adding comments, upload a second patchset "python upload.py –oauth2 -i <issue_ID>"
  - revert the changes on your local repo "hg revert –all"
- 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 http://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 stylus:

```
sudo npm install stylus@0.42.2 -g  # Windows: npm install stylus@0.42.2 -g
# we need 0.42.2 because with more recent versions, --compress compresses
# complete output to 1 line (0.42.2 compresses to 1 line per rule)
stylus -V  # show version number to prove it works
```

- 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
hg diff  # verify nothing changed
```

- check for coding errors (tabs, trailing spaces, line endings, template indentation and spacing):

```
./m coding-std  # Windows: m coding-std
hg diff  # verify nothing changed
```

- check for uncommitted API doc changes:

```
./m api  # Windows m api
hg diff  # verify nothing changed
```

- revert any changes from above:

```
hg revert --all
```

- 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 mercurial, read a tutorial (http://hginit.com/), consider printing a cheatsheet
- if you want a Python IDE, try http://www.jetbrains.com/pycharm/ Free Community Edition
- if you want a graphical interface to Mercurial, install SourceTree (best for mac) or TortoiseHG (best for Windows)
- 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:

```
hg pull -u      # pull all recent changes
./m coding-std  # just in case someone else forgot to do it
./m css         # just in case
hg diff         # expect no changes
./m tests       # note existing errors
```

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 hg diff, hg status - read everything you changed - slowly, look for things that can be improved
  - if you have TortoiseHG or SourceTree, 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 http://jslint.com/

get feedback from other developers

- add changes to codereview: run "python upload.py –oauth2" in your local repo
  - to update a codereview, "python upload.py –oauth2 -i <issue_ID>"
- carefully review your changes again on codereview
  - if you find errors, delete the patchset, fix and upload again
- if you have questions or want to explain something, add comments and click "Publish+Mail Comments"
- post the codereview URL to #moin-dev IRC channel asking for review
- repeat until everybody is happy with your changes

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
```

- pull any changes made by others from the main repo on Bitbucket, merge, then commit the merge
- push the changeset to your public bitbucket repo
- create a pull request so your changes will get pulled into the main repository
- optionally, request a pull on the IRC channel
- 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.

Alternate contribution workflows

If the above workflow looks like overkill (e.g. for simple changes) or you can't work with the tools we usually use, then just create or update an issue on the issue tracker https://bitbucket.org/thomaswaldmann/moin-2.0/issues) or join us on IRC #moin-dev.

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-themes 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, kyoto cabinet/tycoon, mongodb, memory
- jquery javascript lib, a simple jQuery i18n plugin Plugin
- CKeditor, the GUI editor for (x)html
- TWikiDraw, AnyWikiDraw, svgdraw drawing tools

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 *MoinMoin.app.create_app*:

- load the configuration (app.cfg)
- register some modules that handle different parts of the functionality
  - MoinMoin.apps.frontend - most of what a normal user uses
  - MoinMoin.apps.admin - for admins
  - MoinMoin.apps.feed - feeds, e.g. atom
  - MoinMoin.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 *MoinMoin.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 *MoinMoin.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 *MoinMoin.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 *MoinMoin.storage.backends*.

Above that, there is miscellaneous functionality in *MoinMoin.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 *converter.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 *MoinMoin.converter* 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 `MoinMoin/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. Stylesheets for the Modernized and Foobar themes are compiled using the `theme.styl files` in their respective `static/css/stylus` directories. To compile CSS for all themes:

```
./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 `MoinMoin/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");
```

Testing

We use py.test 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 py.test from the toplevel directory:

```
py.test --pep8  # run all tests, including pep8 checks
py.test -rs  # run all tests and output information about skipped tests
py.test -k somekeyword  # run the tests matching somekeyword only
py.test --pep8 -k pep8  # runs pep8 checks only
py.test 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 *py.test* is easy and has little overhead. Just use the *assert* statements.

For more information, please read: http://pytest.org/

Documentation

Sphinx (http://sphinx.pocoo.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
```

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

MoinMoin package

Subpackages

MoinMoin.apps package

Subpackages

MoinMoin.apps.admin package

Submodules

MoinMoin.apps.admin.views module

MoinMoin - admin views

This shows the user interface for wiki admins.

MoinMoin.apps.admin.views.**group_acl_report**(**args*, ***kw*)
Display a 2-column table of items and ACLs, where the ACL rule specifies any WikiGroup or ConfigGroup name.

MoinMoin.apps.admin.views.**groupbrowser**(**args*, ***kw*)
Display list of all groups and their members

MoinMoin.apps.admin.views.**highlighterhelp**()
display a table with list of available Pygments lexers

MoinMoin.apps.admin.views.**index**(**args*, ***kw*)

MoinMoin.apps.admin.views.**index_user**()

MoinMoin.apps.admin.views.**interwikihelp**()
display a table with list of known interwiki names / urls

MoinMoin.apps.admin.views.**item_acl_report**(**args*, ***kw*)
Return a list of all items in the wiki along with the ACL Meta-data

MoinMoin.apps.admin.views.**itemsize**()
display a table with item sizes

MoinMoin.apps.admin.views.**mail_recovery_token**(**args*, ***kw*)
Send user an email so he can reset his password.

MoinMoin.apps.admin.views.**modify_acl**(**args*, ***kw*)

MoinMoin.apps.admin.views.**search_group**(*group_name*)

MoinMoin.apps.admin.views.**trash**(**args*, ***kw*)
Returns the trashed items.

MoinMoin.apps.admin.views.**user_acl_report**(**args*, ***kw*)

MoinMoin.apps.admin.views.**userbrowser**(**args*, ***kw*)
User Account Browser

MoinMoin.apps.admin.views.**userprofile**(**args*, ***kw*)
Set values in user profile

MoinMoin.apps.admin.views.**wikiconfig**(**args*, ***kw*)

MoinMoin.apps.admin.views.**wikiconfighelp**(**args*, ***kw*)

Module contents

MoinMoin - admin views package

This package contains all views, templates, static files for wiki administration.

MoinMoin.apps.feed package

Submodules

MoinMoin.apps.feed.views module

MoinMoin - feed views

This contains all sort of feeds.

MoinMoin.apps.feed.views.**atom**(*item_name*)

Module contents

MoinMoin - feed views package

This package contains all views, templates, static files for feeds (like atom, ...).

MoinMoin.apps.frontend package

Submodules

MoinMoin.apps.frontend.views module

MoinMoin - frontend views

This shows the usual things users see when using the wiki.

MoinMoin.apps.frontend.views.**ContenttypeGroup**
alias of Array

class MoinMoin.apps.frontend.views.**DeleteItemForm**(*value=Unspecified*, ***kw*)
Bases: *MoinMoin.items.BaseChangeForm*

field_schema = [<class 'flatland.schema.declarative.String'>, <class 'flatland.schema.declarative.String'>, <class '

name = 'delete_item'

class MoinMoin.apps.frontend.views.**DestroyItemForm**(*value=Unspecified*, ***kw*)
Bases: *MoinMoin.items.BaseChangeForm*

field_schema = [<class 'flatland.schema.declarative.String'>, <class 'flatland.schema.declarative.String'>, <class '

name = 'destroy_item'

class MoinMoin.apps.frontend.views.**IndexForm**(*value=Unspecified*, ***kw*)
Bases: flatland.schema.declarative.Form

field_schema = [<class 'flatland.schema.declarative.Array'>]

submit_label = l'Filter'

class MoinMoin.apps.frontend.views.**LoginForm**(*value=Unspecified*, ***kw*)
Bases: flatland.schema.declarative.Form

Login form

field_schema = [<class 'flatland.schema.declarative.String'>, <class 'flatland.schema.declarative.String'>, <class '

name = 'login'

submit_label = l'Log in'

validators = [<MoinMoin.apps.frontend.views.ValidLogin object>]

class MoinMoin.apps.frontend.views.**LookupForm**(*value=Unspecified*, ***kw*)
Bases: flatland.schema.declarative.Form

field_schema = [<class 'flatland.schema.declarative.String'>, <class 'flatland.schema.declarative.String'>, <class '

submit_label = l'Lookup'

class MoinMoin.apps.frontend.views.**NestedItemListBuilder**
Bases: object

childs(*fq_name*, *backrefs=False*)

is_ok(*child*)

recurse_build(*fq_names*, *backrefs=False*)

class MoinMoin.apps.frontend.views.**OpenIDForm**(*value=Unspecified*, ***kw*)
Bases: *MoinMoin.apps.frontend.views.RegistrationForm*

OpenID registration form, inherited from the simple registration form.

field_schema = [<class 'flatland.schema.declarative.String'>, <class 'flatland.schema.declarative.String'>, <class '

name = 'openid'

class MoinMoin.apps.frontend.views.**PasswordLostForm**(*value=Unspecified*, ***kw*)
Bases: flatland.schema.declarative.Form

a simple password lost form

field_schema = [<class 'flatland.schema.declarative.String'>, <class 'flatland.schema.declarative.String'>]

name = 'lostpass'

submit_label = l'Recover password'

validators = [<MoinMoin.apps.frontend.views.ValidLostPassword object>]

class MoinMoin.apps.frontend.views.**PasswordRecoveryForm**(*value=Unspecified*, ***kw*)

Bases: flatland.schema.declarative.Form

a simple password recovery form

field_schema = [<class 'flatland.schema.declarative.String'>, <class 'flatland.schema.declarative.String'>, <class '

name = 'recoverpass'

submit_label = l'Change password'

validators = [<MoinMoin.apps.frontend.views.ValidPasswordRecovery object>]

class MoinMoin.apps.frontend.views.**RegistrationForm**(*value=Unspecified*, ***kw*)

Bases: *MoinMoin.security.textcha.TextChaizedForm*

a simple user registration form

field_schema = [<class 'flatland.schema.declarative.String'>, <class 'flatland.schema.declarative.String'>, <class '

name = 'register'

submit_label = l'Register'

validators = [<MoinMoin.apps.frontend.views.ValidRegistration object>]

class MoinMoin.apps.frontend.views.**RenameItemForm**(*value=Unspecified*, ***kw*)

Bases: *MoinMoin.apps.frontend.views.TargetChangeForm*

field_schema = [<class 'flatland.schema.declarative.String'>, <class 'flatland.schema.declarative.String'>, <class '

name = 'rename_item'

class MoinMoin.apps.frontend.views.**RevertItemForm**(*value=Unspecified*, ***kw*)

Bases: *MoinMoin.items.BaseChangeForm*

field_schema = [<class 'flatland.schema.declarative.String'>, <class 'flatland.schema.declarative.String'>, <class '

name = 'revert_item'

validators = [<MoinMoin.apps.frontend.views.ValidRevert object>]

class MoinMoin.apps.frontend.views.**TargetChangeForm**(*value=Unspecified*, ***kw*)

Bases: *MoinMoin.items.BaseChangeForm*

field_schema = [<class 'flatland.schema.declarative.String'>, <class 'flatland.schema.declarative.String'>, <class '

class MoinMoin.apps.frontend.views.**UserSettingsNotificationForm**(*value=Unspecified*, ***kw*)

Bases: flatland.schema.declarative.Form

field_schema = [<class 'flatland.schema.declarative.String'>]

name = 'usersettings_notification'

submit_label = l'Save'

class MoinMoin.apps.frontend.views.**UserSettingsOptionsForm**(*value=Unspecified*, ***kw*)

Bases: flatland.schema.declarative.Form

field_schema = [<class 'flatland.schema.declarative.Boolean'>, <class 'flatland.schema.declarative.Boolean'>, <cla

name = 'usersettings_options'

submit_label = l'Save'

class MoinMoin.apps.frontend.views.**UserSettingsPasswordForm**(*value=Unspecified*, ***kw*)

Bases: flatland.schema.declarative.Form

field_schema = [<class 'flatland.schema.declarative.String'>, <class 'flatland.schema.declarative.String'>, <class '

name = 'usersettings_password'

submit_label = l'Change password'

validators = [<MoinMoin.apps.frontend.views.ValidChangePass object>]

class MoinMoin.apps.frontend.views.**UserSettingsQuicklinksForm**(*value=Unspecified*, ***kw*)

Bases: flatland.schema.declarative.Form

No validation is performed as lots of things are valid, existing items, non-existing items, external links, mailto, external wiki links...

field_schema = [<class 'flatland.schema.declarative.MyJoinedString'>]

name = 'usersettings_quicklinks'

submit_label = l'Save'

class MoinMoin.apps.frontend.views.**UserSettingsSubscriptionsForm**(*value=Unspecified*, ***kw*)

Bases: flatland.schema.declarative.Form

field_schema = [<class 'flatland.schema.declarative.SubscriptionsJoinedString'>]

name = 'usersettings_subscriptions'

submit_label = l'Save'

validators = [<MoinMoin.apps.frontend.views.ValidSubscriptions object>]

class MoinMoin.apps.frontend.views.**ValidChangePass**(***kw*)

Bases: flatland.validation.base.Validator

Validator for a valid password change

current_password_wrong_msg = l'The current password was wrong.'

password_problem_msg = l'New password is unacceptable, could not get processed.'

passwords_mismatch_msg = l'The passwords do not match.'

validate(*element*, *state*)

class MoinMoin.apps.frontend.views.**ValidLogin**(***kw*)

Bases: flatland.validation.base.Validator

Login validator

moin_fail_msg = l'Either your username or password was invalid.'

openid_fail_msg = l'Failed to authenticate with this OpenID.'

validate(*element*, *state*)

class MoinMoin.apps.frontend.views.**ValidLostPassword**(***kw*)

Bases: flatland.validation.base.Validator

Validator for a valid lost password form

name_or_email_needed_msg = l'Your user name or your email address is needed.'

validate(*element*, *state*)

class MoinMoin.apps.frontend.views.**ValidPasswordRecovery**(***kw*)

Bases: flatland.validation.base.Validator

Validator for a valid password recovery form

password_problem_msg = l'New password is unacceptable, could not get processed.'

passwords_mismatch_msg = l'The passwords do not match.'

validate(*element*, *state*)

class MoinMoin.apps.frontend.views.**ValidRegistration**(**kw)
Bases: flatland.validation.base.Validator

Validator for a valid registration form

passwords_mismatch_msg = l'The passwords do not match.'

validate(*element*, *state*)

class MoinMoin.apps.frontend.views.**ValidRevert**(**kw)
Bases: flatland.validation.base.Validator

Validator for a valid revert form.

invalid_name_msg = ‘’

validate(*element*, *state*)
Check whether the names present in the previous meta are not taken by some other item.

class MoinMoin.apps.frontend.views.**ValidSubscriptions**(**kw)
Bases: flatland.validation.base.Validator

Validator for a subscriptions change

validate(*element*, *state*)

MoinMoin.apps.frontend.views.**add_facets**(*facets*, *time_sorting*)
Adds various facets for the search features.

Parameters

- **facets** – current facets
- **time_sorting** – defines the sorting order and can have one of the following 3 values : 1. default - default search 2. old - sort old items first 3. new - sort new items first

Returns required facets for the search query

MoinMoin.apps.frontend.views.**add_file_filters**(*_filter*, *filetypes*)
Add various terms to the filter for the search query for the selected file types in the search options.

Parameters

- **_filter** – the current filter
- **filetypes** – list of selected filetypes

Returns the required _filter for the search query

MoinMoin.apps.frontend.views.**add_presenter**(*wrapped*, *view*, *add_trail=False*, *abort404=True*)
Add new “presenter” views.

Presenter views handle GET requests to locations like +{view}/+<rev>/<item_name> and +{view}/<item_name>, and always try to look up the item before processing.

Parameters

- **view** – name of view
- **add_trail** – whether to call flaskg.user.add_trail
- **abort404** – whether to abort(404) for nonexistent items

MoinMoin.apps.frontend.views.**ajaxdelete**(*item_name*)

MoinMoin.apps.frontend.views.**ajaxdestroy**(*item_name*)

MoinMoin.apps.frontend.views.**ajaxmodify**(*item_name*)

MoinMoin.apps.frontend.views.**analyze**(*analyzer*, *text*)

MoinMoin.apps.frontend.views.**backrefs**(*item_name*)
Returns the list of all items that link or transclude item_name

Parameters **item_name** (*unicode*) – the name of the current item

Returns a page with all the items which link or transclude item_name

MoinMoin.apps.frontend.views.**bookmark**()
set bookmark (in time) for recent changes (or delete them)

MoinMoin.apps.frontend.views.**closeMatches**(*fq_name*, *fq_names*)
Get close matches.

Return all matching fqnames with rank above cutoff value.

Parameters

- **fq_name** – fqname to match
- **fq_names** – list of fqnames

Return type *list*

Returns list of matching item names, sorted by rank

MoinMoin.apps.frontend.views.**comment**(*item_name*)
Initiated by tickets.js when user clicks Save button adding a reply to a prior comment.

An html fragment formatting a new comment is produced. It is inserted into the page via javascript.

MoinMoin.apps.frontend.views.**content_item**(*item_name*, *rev*)
same as show_item, but we only show the content

MoinMoin.apps.frontend.views.**contenttype_selects_gen**()

MoinMoin.apps.frontend.views.**convert_item**(*item_name*)
return a converted item.

We create two items : the original one, and an empty one with the expected mimetype for the converted item.

To get the converted item, we just feed his converter, with the internal representation of the item.

MoinMoin.apps.frontend.views.**delete_item**(*item_name*)

MoinMoin.apps.frontend.views.**destroy_item**(*item_name*, *rev*)

MoinMoin.apps.frontend.views.**diff**(*item_name*)

MoinMoin.apps.frontend.views.**diffraw**(*item_name*)

MoinMoin.apps.frontend.views.**dispatch**()

MoinMoin.apps.frontend.views.**download_item**(*item_name*, *rev*)

MoinMoin.apps.frontend.views.**favicon**()

MoinMoin.apps.frontend.views.**findMatches**(*fq_name*, *s_re=None*, *e_re=None*)
Find similar item names.

Parameters

- **fq_name** – fqname to match
- **s_re** – start re for wiki matching
- **e_re** – end re for wiki matching

Return type tuple

Returns start word, end word, matches dict

MoinMoin.apps.frontend.views.**forwardrefs**(*item_name*)
Returns the list of all links or transclusions of item item_name

Parameters **item_name** (*unicode*) – the name of the current item

Returns a page with all the items linked from this item

MoinMoin.apps.frontend.views.**get_item**(*item_name*, *rev*)

MoinMoin.apps.frontend.views.**get_revs**()
get 2 revids from values

MoinMoin.apps.frontend.views.**global_history**(*namespace*)

MoinMoin.apps.frontend.views.**global_tags**(*namespace*)
show a list or tag cloud of all tags in this wiki

MoinMoin.apps.frontend.views.**global_views**()
Provides a link to all the global views.

MoinMoin.apps.frontend.views.**highlight_item**(*item_name*, *rev*)

MoinMoin.apps.frontend.views.**history**(*item_name*)

MoinMoin.apps.frontend.views.**index**(*item_name*)

MoinMoin.apps.frontend.views.**indexable**(*item_name*, *rev*)

MoinMoin.apps.frontend.views.**jfu_server**(*item_name*)
jquery-file-upload server component

MoinMoin.apps.frontend.views.**login**()

MoinMoin.apps.frontend.views.**logout**()

MoinMoin.apps.frontend.views.**lookup**()
lookup is like search, but it only deals with specific fields that identify an item / revision. no query string parsing.

for uuid fields, it performs a prefix search, so you can just give the first few digits. same is done for name_exact field. if you give a complete uuid or you do a lookup via the name field, it will use a simple search term. for one result, it directly redirects to the item/revision found. for none or multipe results, a result page is shown.

usually this is used for links with a query string, like: /+lookup?itemid=123cba (prefix match on itemid 123cba.....) /+lookup?revid=c0ddcda9a092499c92920cc4a9b11704 (full uuid simple term match) /+lookup?name_exact=FooBar/ (prefix match on name_exact FooBar/...)

When giving history=1 it will use the all revisions index for lookup.

MoinMoin.apps.frontend.views.**lostpass**()

MoinMoin.apps.frontend.views.**modify_item**(*item_name*)
Modify the wiki item item_name.

On GET, displays a form. On POST, saves the new page (unless there's an error in input). After successful POST, redirects to the page.

MoinMoin.apps.frontend.views.**mychanges**()
Returns the list of all items the current user has contributed to.

Returns a page with all the items the current user has contributed to

MoinMoin.apps.frontend.views.**new**()

MoinMoin.apps.frontend.views.**orphaned_items**()
Return a list view of existing items not being linked or transcluded by any other item (which makes them sometimes not discoverable).

MoinMoin.apps.frontend.views.**page_not_found**(*e*)

MoinMoin.apps.frontend.views.**presenter**(*view*, *add_trail=False*, *abort404=True*)
Decorator factory to apply add_presenter().

MoinMoin.apps.frontend.views.**quicklink_item**(*item_name*)
Add/Remove the current wiki page to/from the user quicklinks

MoinMoin.apps.frontend.views.**recoverpass**()

MoinMoin.apps.frontend.views.**redirect_show_item**(*item_name*)

MoinMoin.apps.frontend.views.**register**()

MoinMoin.apps.frontend.views.**rename_item**(*item_name*)

MoinMoin.apps.frontend.views.**revert_item**(*item_name*, *rev*)

MoinMoin.apps.frontend.views.**robots**()

MoinMoin.apps.frontend.views.**search**(*item_name*)

MoinMoin.apps.frontend.views.**show_dom**(*item_name*, *rev*)

MoinMoin.apps.frontend.views.**show_item**(*item_name*, *rev*)

MoinMoin.apps.frontend.views.**show_item_meta**(*item_name*, *rev*)

MoinMoin.apps.frontend.views.**show_root**()

MoinMoin.apps.frontend.views.**similar_names**(*item_name*)
list similar item names

MoinMoin.apps.frontend.views.**sitemap**(*item_name*)
sitemap view shows item link structure, relative to current item

MoinMoin.apps.frontend.views.**split_fqname_list**(*names*)
Converts a list of names to a list of fqnames.

MoinMoin.apps.frontend.views.**subscribe_item**(*item_name*)
Add/Remove the current wiki item to/from the user's subscriptions

MoinMoin.apps.frontend.views.**tagged_items**(*tag*, *namespace*)
show all items' names that have tag <tag> and belong to namespace <namespace>

MoinMoin.apps.frontend.views.**template**(*filename*)
serve a rendered template from <filename>

used for (but not limited to) translation of javascript / css / html

MoinMoin.apps.frontend.views.**ticket_search**()
Suggest duplicate tickets while a new ticket is being created. Executed multiple times as user types/clicks.

TODO: not useful as is, suggestions must match every word in ticket summary. Clicking radio buttons create updates but values seem to have no effect on results. Better suggestions may come from matching on tag values.

MoinMoin.apps.frontend.views.**tickets**()
Show a list of ticket items

MoinMoin.apps.frontend.views.**usersettings**()

MoinMoin.apps.frontend.views.**verifyemail**()

MoinMoin.apps.frontend.views.**wanted_items**()
Returns a list view of non-existing items that are linked to or transcluded by other items. If you want to know by which items they are referred to, use the backrefs functionality of the item in question.

MoinMoin.apps.frontend.views.**wikiMatches**(*fq_name*, *fq_names*, *start_re=None*, *end_re=None*)
Get fqnames that starts or ends with same word as this fq_name.

Matches are ranked like this: 4 - item is subitem of fq_name 3 - match both start and end 2 - match end 1 - match start

Parameters

- **fq_name** – fqname to match
- **fq_names** – list of fqnames
- **start_re** – start word re (compile regex)

- **end_re** – end word re (compile regex)

Return type tuple

Returns start, end, matches dict

Module contents

MoinMoin - frontend views package

This package contains all views, templates, static files that a normal wiki user usually sees.

MoinMoin.apps.misc package

Submodules

MoinMoin.apps.misc.views module

MoinMoin - miscellaneous views

Misc. stuff that doesn't fit into another view category.

MoinMoin.apps.misc.views.**sitemap**()
Google (and others) XML sitemap

MoinMoin.apps.misc.views.**urls_names**()
List of all item URLs and names, e.g. for sisteritems.

This view generates a list of item URLs and item names, so that other wikis can implement SisterWiki functionality easily. See: http://usemod.com/cgi-bin/mb.pl?SisterSitesImplementationGuide

Module contents

MoinMoin - misc. views package

This package contains misc. stuff that doesn't fit into another view category.

MoinMoin.apps.serve package

Submodules

MoinMoin.apps.serve.views module

MoinMoin - external static file serving

MoinMoin.apps.serve.views.**files**(*name*, *filename*)

MoinMoin.apps.serve.views.**index**()

Module contents

MoinMoin - serve (external) static files

E.g. javascript based drawing or html editors. We want to avoid bundling them, thus we access them somewhere on the filesystem outside of moin.

Module contents

MoinMoin - flask modules for better modularization

This package contains some Flask Modules:

- frontend has all usual wiki user interface code
- feed Module for all feed-like stuff
- admin Module for special stuff for wiki admins
- serve Module for static file serving

MoinMoin.auth package

Submodules

MoinMoin.auth.http module

MoinMoin - http authentication

HTTPAuthMoin

HTTPAuthMoin is HTTP auth done by moin (not by your web server).

Moin will request HTTP Basic Auth and use the HTTP Basic Auth header it receives to authenticate username/password against the moin user profiles.

from MoinMoin.auth.http import HTTPAuthMoin auth = [HTTPAuthMoin()]

class MoinMoin.auth.http.**HTTPAuthMoin**(*autocreate=False*, *realm='MoinMoin'*, *coding='iso-8859-1'*, ***kw*)

Bases: *MoinMoin.auth.BaseAuth*

authenticate via http (basic) auth

name = 'http'

request(*user_obj*, ***kw*)

MoinMoin.auth.ldap_login module

MoinMoin.auth.log module

MoinMoin - logging auth plugin

This does nothing except logging the auth parameters (the password is NOT logged, of course).

class MoinMoin.auth.log.**AuthLog**(***kw*)

Bases: *MoinMoin.auth.BaseAuth*

just log the call, do nothing else

log(*action*, *user_obj*, *kw*)

login(*user_obj*, ***kw*)

logout(*user_obj*, ***kw*)

name = 'log'

request(*user_obj*, ***kw*)

MoinMoin.auth.openidrp module

MoinMoin.auth.smb_mount module

MoinMoin - auth plugin for (un)mounting a smb share

(u)mount a SMB server's share for username (using username/password for authentication at the SMB server). This can be used if you need access to files on some share via the wiki, but needs more code to be useful.

class MoinMoin.auth.smb_mount.**SMBMount**(*server*, *share*, *mountpoint_fn*, *dir_user*, *domain*, *dir_mode='0700'*, *file_mode='0600'*, *iocharset='utf-8'*, *coding='utf-8'*, *log='/dev/null'*, ***kw*)

Bases: *MoinMoin.auth.BaseAuth*

auth plugin for (un)mounting an smb share, this is a wrapper around mount.cifs -o <options> //server/share mountpoint

See man mount.cifs for details.

do_smb(*username*, *password*, *login*)

login(*user_obj*, ***kw*)

logout(*user_obj*, ***kw*)

Module contents

MoinMoin - modular authentication handling

Each authentication method is an object instance containing four methods:

- `login(user_obj, **kw)`
- `logout(user_obj, **kw)`
- `request(user_obj, **kw)`
- `login_hint()`

The kw arguments that are passed in are currently:

attended: boolean indicating whether a user (attended=True) or a machine is requesting login, multistage auth is not currently possible for machine logins [login only]

username: the value of the 'username' form field (or None) [login only]

password: the value of the 'password' form field (or None) [login only]

cookie: a Cookie.SimpleCookie instance containing the cookie that the browser sent

multistage: boolean indicating multistage login continuation [may not be present, login only]

login_hint() should return a HTML text that is displayed to the user right below the login form, it should tell the user what to do in case of a forgotten password and how to create an account (if applicable.)

More may be added.

The request method is called for each request except login/logout.

The 'request' and 'logout' methods must return a tuple (user_obj, continue) where 'user_obj' can be:

- None, to throw away any previous user_obj from previous auth methods
- the passed in user_obj for no changes
- a newly created MoinMoin.user.User instance

and 'continue' is a boolean to indicate whether the next authentication method should be tried.

The 'login' method must return an instance of MoinMoin.auth.LoginReturn which contains the members:

- user_obj
- continue_flag
- multistage
- message
- redirect_to

There are some helpful subclasses derived from this class for the most common cases, namely ContinueLogin(), CancelLogin(), MultistageFormLogin() and MultistageRedirectLogin().

The user_obj and continue_flag members have the same semantics as for the request and logout methods.

The messages that are returned by the various auth methods will be displayed to the user, since they will all be displayed usually auth methods will use the message feature only along with returning False for the continue flag.

Note, however, that when no username is entered or the username is not found in the database, it may be appropriate to return with a message and the continue flag set to true (ContinueLogin) because a subsequent auth plugin might work even without the username (e.g. an openid auth plugin).

The multistage member must evaluate to false or be callable. If it is callable, this indicates that the authentication method requires a second login stage. In that case, the multistage item will be called and should return an instance of MoinMoin.widget.html.FORM and the generic code will append some required hidden fields to it. It is also permissible to return some valid HTML, but that feature has very limited use since it breaks the authentication method chain.

Note that because multistage login does not depend on anonymous session support, it is possible that users jump directly into the second stage by giving the appropriate parameters to the login action. Hence, auth methods should take care to recheck everything and not assume the user has gone through all previous stages.

If the multistage login requires querying an external site that involves a redirect, the redirect_to member may be set instead of the multistage member. If this is set it must be a URL that user should be redirected to. Since the user must be able to come back to the authentication, any "%return" in the URL is replaced with the url-encoded form of the URL to the next authentication stage, any "%return_form" is replaced with the url-plus-encoded form (spaces encoded as +) of the same URL.

After the user has submitted the required form or has been redirected back from the external site, execution of the auth login methods resumes with the auth item that requested the multistage login and its login method is called with the 'multistage' keyword parameter set to True.

Each authentication method instance must also contain the members:

- **login_inputs: a list of required inputs, currently supported are**
  - 'username': username entry field
  - 'password': password entry field
  - **'special_no_input': manual login is required** but no form fields need to be filled in (e.g. openid with forced provider) in this case the theme may provide a short- cut omitting the login form
- **logout_possible: boolean indicating whether this auth methods** supports logging out
- **name: name of the auth method, must be the same as given as the** user object's auth_method keyword parameter.

To simplify creating new authentication methods you can inherit from MoinMoin.auth.BaseAuth that does nothing for all three methods, but allows you to override only some methods.

cfg.auth is a list of authentication object instances whose methods are called in the order they are listed. The session method is called for every request, when logging in or out these are called before the session method.

When creating a new MoinMoin.user.User object, you can give a keyword argument "auth_attribs" to User.__init__ containing a list of user attributes that are determined and fixed by this auth method and may not be changed by the user in their preferences. You also have to give the keyword argument "auth_method" containing the name of the authentication method.

class MoinMoin.auth.**BaseAuth**(*trusted=False*, ***kw*)
Bases: object

login(*user_obj*, ***kw*)

login_hint()

login_inputs = []

logout(*user_obj*, ***kw*)

logout_possible = False

name = None

request(*user_obj*, ***kw*)

class MoinMoin.auth.**CancelLogin**(*message*)
Bases: *MoinMoin.auth.LoginReturn*

CancelLogin - cancel login showing a message

class MoinMoin.auth.**ContinueLogin**(*user_obj*, *message=None*)
Bases: *MoinMoin.auth.LoginReturn*

ContinueLogin - helper for auth method login that just continues

class MoinMoin.auth.**GivenAuth**(*env_var=None*, *user_name=None*, *autocreate=False*, *strip_maildomain=False*, *strip_windomain=False*, *titlecase=False*, *remove_blanks=False*, *coding='utf-8'*, ***kw*)
Bases: *MoinMoin.auth.BaseAuth*

reuse a given authentication, e.g. http basic auth (or any other auth) done by the web server, that sets REMOTE_USER environment variable. This is the default behaviour. You can also specify to read another environment variable (env_var). Alternatively you can directly give a fixed user name (user_name) that will be considered as authenticated.

decode_username(*name*)
decode the name we got from the environment var to unicode

name = 'given'

request(*user_obj*, ***kw*)

transform_username(*name*)
transform the name we got (unicode in, unicode out)

Note: if you need something more special, you could create your own auth class, inherit from this class and overwrite this function.

class MoinMoin.auth.**LoginReturn**(*user_obj*, *continue_flag*, *message=None*, *multistage=None*, *redirect_to=None*)
Bases: object

LoginReturn - base class for auth method login() return value

class MoinMoin.auth.**MoinAuth**(***kw*)
Bases: *MoinMoin.auth.BaseAuth*

handle login from moin login form

login(*user_obj*, ***kw*)

login_hint()

login_inputs = ['username', 'password']

logout_possible = True

name = 'moin'

class MoinMoin.auth.**MultistageFormLogin**(*multistage*)
Bases: *MoinMoin.auth.LoginReturn*

MultistageFormLogin - require user to fill in another form

class MoinMoin.auth.**MultistageRedirectLogin**(*url*)
Bases: *MoinMoin.auth.LoginReturn*

MultistageRedirectLogin - redirect user to another site before continuing login

MoinMoin.auth.**get_multistage_continuation_url**(*auth_name*, *extra_fields={}*)
get_continuation_url - return a multistage continuation URL

This function returns a URL that when loaded continues a multistage authentication at the auth method requesting it (parameter auth_name.) Additional fields are added to the URL from the extra_fields dict.

Parameters

- **auth_name** – name of the auth method requesting the continuation
- **extra_fields** – extra GET fields to add to the URL

MoinMoin.auth.**handle_login**(*userobj*, ***kw*)
Process a 'login' request by going through the configured authentication methods in turn. The passable keyword arguments are explained in more detail at the top of this file.

MoinMoin.auth.**handle_logout**(*userobj*)
Logout the passed user from every configured authentication method.

MoinMoin.auth.**handle_request**(*userobj*)
Handle the per-request callbacks of the configured authentication methods.

MoinMoin.auth.**setup_from_session**()

MoinMoin.config package

Submodules

MoinMoin.config.default module

MoinMoin - Configuration defaults class

class MoinMoin.config.default.**CacheClass**
Bases: object

just a container for stuff we cache

class MoinMoin.config.default.**ConfigFunctionality**
Bases: object

Configuration base class with config class behaviour.

This class contains the functionality for the DefaultConfig class for the benefit of the WikiConfig macro.

auth_can_logout = None

auth_have_login = None

auth_login_inputs = None

cache = None

mail_enabled = None

siteid = None

class MoinMoin.config.default.**DefaultConfig**

Bases: *MoinMoin.config.default.ConfigFunctionality*

Configuration base class with default config values (added below)

SecurityPolicy
alias of DefaultSecurityPolicy

acl_functions = u''

acl_mapping = None

acl_rights_contents = ['read', 'pubread', 'write', 'create', 'admin', 'destroy']

acl_rights_functions = ['superuser', 'notextcha']

allow_style_attributes = False

auth = [<MoinMoin.auth.MoinAuth object>]

backend_mapping = None

bang_meta = True

config_check_enabled = False

create_index = False

create_storage = False

data_dir = './data/'

default_root = u'Home'

destroy_index = False

destroy_storage = False

dicts(*cfg*)

edit_ticketing = True

endpoints_excluded = []

groups(*cfg*)

html_pagetitle = None

interwiki_map = {}

interwiki_preferred = []

interwikiname = None

item_dict_regex = u'(?P<all>(?P<key>\\S+)Dict)'

item_group_regex = u'(?P<all>(?P<key>\\S+)Group)'

item_license = u''

item_views = [('frontend.show_item', l'Show', l'Show', False), ('frontend.download_item', l'Download', l'Downloa

locale_default = u'en_US'

log_remote_addr = True

log_reverse_dns_lookups = True

mail_from = None

mail_password = None

mail_sendmail = None

mail_smarthost = None

mail_username = None

mimetypes_to_index_as_empty = []

mimetypes_xss_protect = ['text/html', 'application/x-shockwave-flash', 'application/xhtml+xml']

namespace_mapping = None

navi_bar = [('wikilink', 'frontend.show_root', {}, l'Home', l'Home Page'), ('wikilink', 'frontend.global_history', {},

ns_content = '/'

ns_user_homepage = 'User/'

ns_user_profile = 'UserProfile/'

passlib_crypt_context = {'schemes': ['sha512_crypt']}

password_checker(*cfg*, *username*, *password*, *min_length=8*, *min_different=5*)

Check if a password is secure enough. We use a built-in check to get rid of the worst passwords.

We do NOT use cracklib / python-crack here any more because it is not thread-safe (we experienced segmentation faults when using it).

If you don't want to check passwords, use password_checker = None.

Returns None if there is no problem with the password, some unicode object with an error msg, if the password is problematic.

plugin_dirs = []

refresh = None

results_per_page = 50

root_mapping = {}

secrets = None

serve_files = {}

show_hosts = True

show_interwiki = False

show_names = True

show_rename_redirect = False

show_section_numbers = False

sistersites = []

siteid = 'MoinMoin'

sitename = u'Untitled Wiki'

supplementation_item_names = [u'Discussion']

template_dirs = []

textchas = None

textchas_expiry_time = 600

theme_default = u'topside'

timezone_default = u'UTC'

trail_size = 5

user_defaults = {u'email_unvalidated': None, u'theme_name': None, u'locale': None, u'disabled': False, u'quick

user_email_unique = True

user_email_verification = False

user_homewiki = u'Self'

user_use_gravatar = False

class MoinMoin.config.default.**DefaultExpression**(*exprstr*)
Bases: object

Module contents

MoinMoin.constants package

Submodules

MoinMoin.constants.chartypes module

MoinMoin.constants.contenttypes module

MoinMoin - contenttype related constants

MoinMoin.constants.contenttypes.**ext_link**(*href*, *link_text=None*)

MoinMoin.constants.forms module

MoinMoin - Flatland form related constants

MoinMoin.constants.itemtypes module

MoinMoin - itemtype related constants

MoinMoin.constants.keys module

MoinMoin - meta data key / index field name related constants

MoinMoin.constants.misc module

MoinMoin - misc. constants not fitting elsewhere

MoinMoin.constants.namespaces module

MoinMoin - namespaces related constants

MoinMoin.constants.rights module

MoinMoin - ACL related constants

Module contents

MoinMoin - modules with constant definitions

MoinMoin.converter package

Submodules

MoinMoin.converter.archive_in module

MoinMoin - Archives converter (e.g. zip, tar)

Make a DOM Tree representation of an archive (== list contents of it in a table).

class MoinMoin.converter.archive_in.**ArchiveConverter**
Bases: MoinMoin.converter._table.TableMixin

Base class for archive converters, convert an archive to a DOM table with an archive listing.

list_contents(*fileobj*)
analyze archive we get as fileobj and return data for table rendering.

We return a list of rows, each row is a list of cells.

Usually each row is [size, datetime, name] for each archive member.

In case of problems, it shall raise ArchiveException(error_msg).

process_datetime(*dt*)

process_name(*member_name*)

process_size(*size*)

exception MoinMoin.converter.archive_in.**ArchiveException**
Bases: exceptions.Exception

exception class used in case of trouble with opening/listing an archive

class MoinMoin.converter.archive_in.**TarConverter**
Bases: *MoinMoin.converter.archive_in.ArchiveConverter*

Support listing tar files.

list_contents(*fileobj*)

class MoinMoin.converter.archive_in.**ZipConverter**
Bases: *MoinMoin.converter.archive_in.ArchiveConverter*

Support listing zip files.

list_contents(*fileobj*)

MoinMoin.converter.audio_video_in module

MoinMoin - Audio/Video converter

Convert audio/video to <object> tag for the DOM Tree.

Note: currently this is quite same as image_in.

class MoinMoin.converter.audio_video_in.**Converter**(*input_type*)
Bases: object

Convert audio/video to the corresponding <object> in the DOM Tree

MoinMoin.converter.creole_in module

MoinMoin - Creole input converter

See http://wikicreole.org/ for latest specs.

Notes:

- No markup allowed in headings. Creole 1.0 does not require us to support this.
- No markup allowed in table headings. Creole 1.0 does not require us to support this.
- No (non-bracketed) generic url recognition: this is "mission impossible" except if you want to risk lots of false positives. Only known protocols are recognized.
- We do not allow ":" before "//" italic markup to avoid urls with unrecognized schemes (like wtf://server/path) triggering italic rendering for the rest of the paragraph.

class MoinMoin.converter.creole_in.**Converter**
Bases: MoinMoin.converter._wiki_macro.ConverterMacro

block = ('(?P<line> ^ \\s* $)', '\n (?P<head>\n ^\n \\s*\n (?P<head_head> =+)\n \\s*\n (?P<head_text> .*?)\n \\s*\n =

block_head = '\n (?P<head>\n ^\n \\s*\n (?P<head_head> =+)\n \\s*\n (?P<head_text> .*?)\n \\s*\n =*\n \\s*\n $\n)

block_head_repl(*_iter_content*, *stack*, *head*, *head_head*, *head_text*)

block_line = '(?P<line> ^ \\s* $)'

block_line_repl(*_iter_content*, *stack*, *line*)

block_list = '\n (?P<list>\n ^\n \\s*\n [*\\#][^*\\#]\n .*\n $\n)\n '

block_list_repl(*iter_content*, *stack*, *list*)

block_macro = '\n ^\n \\s*\n (?P<macro>\n <<\n (?P<macro_name> \\w+)\n (\n \\(\n (?!.*>>.*>>)\n (?P<macro_args

block_macro_repl(*_iter_content*, *stack*, *macro*, *macro_name*, *macro_args=None*)
Handles macros using the placeholder syntax.

block_nowiki = '\n (?P<nowiki>\n ^{{{\n \\s*\n $\n)\n '

block_nowiki_lines(*iter_content*)
Unescaping generator for the lines in a nowiki block

block_nowiki_repl(*iter_content*, *stack*, *nowiki*)
Handles a complete nowiki block

block_re = <_sre.SRE_Pattern object at 0x45b67c0>

block_separator = '(?P<separator> ^ \\s* —- \\s* $)'

block_separator_repl(*_iter_content*, *stack*, *separator*, *hr_class=u'moin-hr3'*)

block_table = '\n (?P<table>\n ^ \\s* \\| .* $\n)\n '

block_table_repl(*iter_content*, *stack*, *table*)

block_table_row(*content*, *stack*)

block_text = '(?P<text> .+)'

block_text_repl(*_iter_content*, *stack*, *text*)

classmethod factory(*input*, *output*, ***kw*)

inline = ('\n (?P<url>\n (^ | (?<=\\s | [.,:;!?()/=]))\n (?P<escaped_url>~)?\n (?P<url_target>\n (http|https|ftp|file|mai

inline_emph = '(?P<emph> (?<!:)//)'

inline_emph_repl(*stack*, *emph*)

inline_escape = '(?P<escape> ~ (?P<escaped_char>\\S))'

inline_escape_repl (*stack*, *escape*, *escaped_char*)

inline_insert = '(?P<insert> (?<!:)__)'

inline_insert_repl (*stack*, *insert*)

inline_linebreak = '(?P<linebreak> \\\\\\\\ )'

inline_linebreak_repl (*stack*, *linebreak*)

inline_link = '\n (?P<link>\n \\[\\[\n \\s*\n (\n (?P<link_url>\n (http|https|ftp|file|mailto|nntp|news|ssh|telnet|irc|irc

inline_link_repl (*stack*, *link*, *link_url=None*, *link_item=None*, *link_text=None*, *link_interwiki_site=None*, *link_interwiki_item=None*)
Handle all kinds of links.

inline_macro = '\n (?P<macro>\n <<\n (?P<macro_name> \\w+)\n (?:\n \\(\n (?P<macro_args> .*?)\n \\)\n)?\n \\s*

inline_macro_repl (*stack*, *macro*, *macro_name*, *macro_args=None*)
Handles macros using the placeholder syntax.

inline_nowiki = '\n (?P<nowiki>\n {{{\n (?P<nowiki_text>.*?}*)\n }}}\n)\n '

inline_nowiki_repl (*stack*, *nowiki*, *nowiki_text*)

inline_object = '\n (?P<object>\n {{\n \\s*\n (\n (?P<object_url>\n [a-zA-Z0-9+.-]+\n ://\n [^|]+?\n)\n |\n (?P<obj

inline_object_repl (*stack*, *object*, *object_page=None*, *object_url=None*, *object_text=None*)
Handles objects included in the page.

inline_re = <_sre.SRE_Pattern object at 0x45b9f70>

inline_strong = '(?P **)'

inline_strong_repl (*stack*, *strong*)

inline_url = '\n (?P<url>\n (^ | (?<=\\s | [.,;;!?()/=]))\n (?P<escaped_url>~)?\n (?P<url_target>\n (http|https|ftp|file

inline_url_repl (*stack*, *url*, *url_target*, *escaped_url=None*)
Handle raw urls in text.

inlinedesc = ('\n (?P<macro>\n <<\n (?P<macro_name> \\w+)\n (?:\n \\(\n (?P<macro_args> .*?)\n \\)\n)?\n \\s*\n

inlinedesc_re = <_sre.SRE_Pattern object at 0x45bbd00>

link_desc = ('\n (?P<object>\n {{\n \\s*\n (\n (?P<object_url>\n [a-zA-Z0-9+.-]+\n ://\n [^|]+?\n)\n |\n (?P<object_p

link_desc_re = <_sre.SRE_Pattern object at 0x45bb490>

list = ('\n (?P<end>\n ^\n (\n # End the list on blank line,\n $\n |\n # heading,\n =\n |\n # table,\n \\|\n |\n # and nowik

list_end = '\n (?P<end>\n ^\n (\n # End the list on blank line,\n $\n |\n # heading,\n =\n |\n # table,\n \\|\n |\n # and n

list_end_repl (*_iter_content*, *stack*, *end*)

list_item = '\n (?P<item>\n ^\n \\s*\n (?P<item_head> [\\#*]+)\n \\s*\n (?P<item_text> .*?)\n $\n)\n '

list_item_repl (*_iter_content*, *stack*, *item*, *item_head*, *item_text*)

list_re = <_sre.SRE_Pattern object at 0x45a8cc0>

list_text = '(?P<text> .+)'

list_text_repl (*_iter_content*, *stack*, *text*)

macro_text (*text*)
Return an ET tree branch representing the markup present in the input text. Used for FootNotes, etc.

nowiki_end = '\n ^ (?P<escape> ~)? (?P<rest> }}} \\s*) $\n '

nowiki_end_re = <_sre.SRE_Pattern object>

**nowiki_interpret = '\n ^\n \\#!\n \\s*\n (?P<nowiki_name> [\\w/]+)?\n \\s*\n (:?\n \\(\n (?P<nowiki_args> .*?)\n **

nowiki_interpret_re = <_sre.SRE_Pattern object>

parse_block(*iter_content*, *arguments*)

parse_inline(*text*, *stack*, *inline_re=<_sre.SRE_Pattern object at 0x45b9f70>*)
Recognize inline elements within the given text

table = '\n (?P<table>\n ^ \\s* \\| .* $\n)\n '

table_re = <_sre.SRE_Pattern object>

tablerow = '\n (?P<cell>\n \\|\n \\s*\n (?P<cell_head> [=])?\n (?P<cell_text> [^|]+)\n \\s*\n)\n '

tablerow_cell_repl(*stack*, *cell*, *cell_text*, *cell_head=None*)
Creole has feature that allows table headings to be either row based or column based.

We avoid use of HTML5 row based thead tag and apply CSS styling to any cell marked as a heading.

tablerow_re = <_sre.SRE_Pattern object>

MoinMoin.converter.docbook_in module

MoinMoin - DocBook input converter Converts a DocBook document into an internal document tree.

Currently supports DocBook v5.

Some elements of DocBook v4 specification are also supported for backward compatibility:

- ulink

class MoinMoin.converter.docbook_in.**Converter**
Bases: object

Converter application/docbook+xml -> x.moin.document

admonition_tags = set(['danger', 'hint', 'attention', 'tip', 'note', 'important', 'caution', 'error', 'warning'])

block_tags = set(['set', 'subtitle', 'simplesect', 'figure', 'cmdsynopsis', 'synopfragment', 'taskrelated', 'sidebar', 'c

do_children(*element*, *depth*)
Function to process the conversion of the children of a given element.

docbook_namespace = {<Namespace('http://docbook.org/ns/docbook')>: 'docbook'}

error(*message*)
Return a DOM Tree containing an error message.

get_standard_attributes(*element*)
We will extract the standard attributes of the element, if any. We save the result in our standard attribute.

ignored_tags = set(['refpurpose', 'itermset', 'otheraddr', 'citerefentry', 'refclass', 'funcsynopsis', 'refsynopsisdivt

inline_tags = set(['constant', 'keycombo', 'errorname', 'package', 'hardware', 'affiliation', 'street', 'termdef', 'po

media_tags = {'audioobject': (['x-wav', 'mpeg', 'ogg', 'webm'], 'audiodata', 'audio/'), 'videoobject': (['ogg', 'webn

new(*tag*, *attrib*, *children*)
Return a new element for the DocBook Tree.

new_copy(*tag*, *element*, *depth*, *attrib*)
Function to copy one element to the DocBook Tree.

It first converts the children of the element, and then the element itself.

root_tags = set(['sect4', 'blockquote', 'sect2', 'sect3', 'sect1', 'section', 'formalpara', 'informalfigure', 'segmentedli

sect_re = <_sre.SRE_Pattern object>

simple_tags = {'programlisting': Name(u'blockcode', u'http://moinmo.in/namespaces/page'), 'code': Name(u'cod

start_dom_tree(*element*, *depth*)
Return the root element of the DOM tree, with all the children.

We also add a <table-of-content> element if needed.

visit(*element*, *depth*)
Function called at each element, to process it.

It will just determine the namespace of our element, then call a dedicated function to handle conversion for the given namespace.

visit_data_element(*element*, *depth*, *object_data*, *text_object*, *caption*)
We will try to return an object element based on the object_data. If it is not possible, we return a paragraph with the content of text_object.

visit_data_object(*element*, *depth*)
Process a mediaobject element. Possible child tags are videoobject, audioobject, imageobject, caption, objectinfo, and textobject.

<mediaobject><videoobject><videodata fileref="video.mp4"/></videoobject></mediaobject>

visit_docbook(*element*, *depth*)
Function called to handle the conversion of DocBook elements to the moin_page DOM Tree.

We will detect the name of the tag, and pick up the correct method to convert it.

visit_docbook_admonition(*element*, *depth*)
<tag.name> –> <admonition type='tag.name'>

visit_docbook_block(*element*, *depth*)
Convert a block element which does not have equivalence in the DOM Tree.

<tag.name> –>

visit_docbook_blockquote(*element*, *depth*)

<blockquote> <attribution>Author</attribution> Text

</blockquote> –> <blockquote source="Author">Text</blockquote>

<blockquote>Text</blockquote> –> <blockquote source="Unknow">Text</blockquote>

visit_docbook_emphasis(*element*, *depth*)
emphasis element, is the only way to apply some style on a DocBook element directly from the DocBook tree.

Basically, you can use it for "italic" and "bold" style.

However, it is still semantic, so we call it emphasis and strong.

visit_docbook_entry(*element*, *depth*)
<td> –> <table-cell>

visit_docbook_entrytbl(*element*, *depth*)
Return a table within a table-cell.

visit_docbook_footnote(*element*, *depth*)
<footnote> –> <note note-class="footnote"><note-body>

visit_docbook_formalpara(*element*, *depth*)

<formalpara> <title>Heading</title> <para>Text</para>

</formalpara> –> Text

visit_docbook_informalequation(*element*, *depth*)
<informalequation> –>

visit_docbook_informalexample(*element*, *depth*)
<informalexample> –>

visit_docbook_informalfigure(*element*, *depth*)
<informalfigure> –>

visit_docbook_inline(*element*, *depth*)
For some specific tags (defined in inline_tags) We just return

visit_docbook_inlinemediaobject(*element*, *depth*)

visit_docbook_inlinequation(*element*, *depth*)
<inlinequation> –>

visit_docbook_itemizedlist(*element*, *depth*)
<itemizedlist> –> <list item-label-generate="unordered">

visit_docbook_link(*element*, *depth*)
LINK Conversion.

There is two kind of links in DocBook : One using the xlink namespace. The other one using linkend attribute.

The xlink attributes can directly be used in the <a> tag of the DOM Tree since we support xlink.

For the linkend attribute, we need to have a system supporting the anchors.

visit_docbook_literallayout(*element*, *depth*)
<literallayout> –> <blockcode html:class="db-literallayout">

visit_docbook_mediaobject(*element*, *depth*)

visit_docbook_olink(*element*, *depth*)

<olink targetdoc='URI' targetptr='ptr'> –> <a xlink:href='URI#ptr'>

visit_docbook_orderedlist(*element*, *depth*)
<orderedlist> –> <list item-label-generate="ordered"> See attribute_conversion for more details about the attributes.

visit_docbook_procedure(*element*, *depth*)
<procedure> –> <list item-label-generate="ordered">

visit_docbook_qandaset(*element*, *depth*)
See visit_qandaset_* method.

visit_docbook_sbr(*element*, *depth*)
<sbr /> –> <line-break />

visit_docbook_sect(*element*, *depth*)
This is the function to convert a numbered section.

Numbered section uses tag like <sectN> where N is the number of the section between 1 and 5.

The sections are supposed to be correctly nested.

We only convert a section to an heading if one of the children is a title element.

TODO: See if we can unify with recursive section below. TODO: Add div element, with specific id

visit_docbook_section(*element*, *depth*)
This is the function to convert recursive section.

Recursive section use tag like <section> only.

Each section, inside another section is a subsection.

To convert it, we will use the depth of the element, and two attributes of the converter which indicate the current depth of the section and the current level heading.

visit_docbook_seglistitem(*element*, *labels*, *depth*)
A seglistitem is a list-item for a segmented list. It is quite special because it act list definition with label, but the labels are predetermined in the labels list.

So we generate label/body couple according to the content in labels

visit_docbook_segmentedlist (*element*, *depth*)
A segmented list is a like a list of definition, but the label are defined at the start with <segtitle> tag and then for each definition, we repeat the label.

So to convert such list, we will first determine and save the labels. Then we will iterate over the object to get the definition.

visit_docbook_simplelist (*element*, *depth*)
<simplelist> –> <list item-label-generate="unordered">

visit_docbook_subscript (*element*, *depth*)
<subscript> –>

visit_docbook_substeps (*element*, *depth*)
Return the same elements than a procedure

visit_docbook_superscript (*element*, *depth*)
<superscript> –>

visit_docbook_table (*element*, *depth*)
<table> –> <table>

visit_docbook_tag (*element*, *depth*)

<tag class="class.name" namespace="ns.address">TAG</tag> –> {ns.address}TAG</tag>

visit_docbook_trademark (*element*, *depth*)
Depending of the trademark class, a specific entities is added to the string.

Docbook supports 4 types of trademark: copyright, registered, trade (mark), and service (mark). <trademark> –>

visit_docbook_ulink (*element*, *depth*)
NB : <ulink> is not a part of DocBook v.5 however we support it in our converter since it is still widely used and it helps to keep a compatibility with DocBook v.4

visit_qandaentry_number (*element*, *depth*)
Convert:

```
<question>Q</question><answer>A</answer>
```

to:

```
<list-item>
    <list-item-body><p>Q</p><p>A</p></list-item-body>
</list-item>
```

visit_qandaentry_qanda (*element*, *depth*)
Convert:

```
<question>Q body</question><answer>A Body</answer>
```

to:

```
<list-item>
    <list-item-label>Q:</list-item-label>
    <list-item-body>Q Body</list-item-body>
</list-item>
<list-item>
    <list-item-label>A:</list-item-label>
    <list-item-body>A Body</list-item-body>
</list-item>
```

visit_qandaset_number (*element*, *depth*)

<qandaset defaultlabel="number"> –> <list item-label-generate='ordered'>

visit_qandaset_qanda(*element*, *depth*)
<qandaset defaultlabel="qanda"> –> <list>

visit_simple_list(*moin_page_tag*, *attrib*, *element*, *depth*)
There is different list element in DocBook with different semantic meaning, but with an unique result in the DOM Tree.

Here we handle the conversion of such of list.

visit_simple_tag(*element*, *depth*)
Some docbook tags can be converted directly to an equivalent DOM Tree element. We retrieve the equivalent tag from the simple_tags dictionnary defined at the begining of this file.

exception MoinMoin.converter.docbook_in.**NameSpaceError**
Bases: exceptions.Exception

MoinMoin.converter.docbook_in.**XML**(*text*, *parser=None*)
Copied from EmeraldTree/tree.py to force use of local XMLParser class override.

class MoinMoin.converter.docbook_in.**XMLParser**(*html=0*, *target=None*, *encoding=None*)
Bases: emeraldtree.tree.XMLParser

Override EmeraldTree/tree.py XMLParser class. Required to add auto-scroll textarea feature.

There is no need to subclass all tree.py classes and procedures with stubs because this modified _start_list is only needed for the initial construction of the DOM when flaskg.add_lineno_attr may be True.

MoinMoin.converter.docbook_out module

MoinMoin - DocBook output converter

Converts an internal document tree into a DocBook v5 document.

class MoinMoin.converter.docbook_out.**Converter**
Bases: object

Converter application/x.moin.document -> application/docbook+xml

admonition_tags = set(['note', 'tip', 'warning', 'important', 'caution'])

do_children(*element*)
Function to process the conversion of the children of a given element.

Return a list of elements.

get_standard_attributes(*element*)
We will extract the standard attributes of the element, if any. We save the result in standard_attribute.

handle_simple_list(*docbook_tag*, *element*, *attrib*)

namespaces_visit = {<Namespace('http://moinmo.in/namespaces/page')>: 'moinpage'}

new(*tag*, *attrib*, *children*)
Return a new element in the DocBook tree.

new_copy(*tag*, *element*, *attrib*)
Function to copy one element to the DocBook tree

It first converts the children of the element, and then the element itself

simple_tags = {'list-item': Name(u'varlistentry', u'http://docbook.org/ns/docbook'), 'emphasis': Name(u'emphasi

standard_attribute = {}

unsupported_tags = set(['separator'])

visit(*element*)
Function called at each element to process it.

It will just determine the namespace of our element, then call a dedicated function to handle conversion for the found namespace.

visit_moinpage(*element*)
Function called to handle the conversion of elements belonging to the moin_page namespace.

We will choose the most appropriate procedure to convert the element according to the tag name

visit_moinpage_a(*element*)
LINK Conversion.

Link are defined using the XLINK namespace either for the DOM Tree and in DocBook specification, so the converter can just copy each xlink: attribute into an <link> tag.

visit_moinpage_admonition(*element*)
There is 5 admonition in DocBook, which are also supported in the DOM Tree.

For instance: <caution> –> <admonition type='caution'>

visit_moinpage_blockcode(*element*)
<blockcode>text</blockcode> –> <screen><![CDATA[text]]></scren>

visit_moinpage_blockquote(*element*)
Convert:

```
<blockquote>text<blockquote>
```

to:

```
<blockquote>
    <attribution>Unknown</attribution>
    <simpara>text</text>
</blockquote>
```

Expand:

```
<blockquote source="author">text</blockquote>
```

output:

```
<blockquote>
    <attribution>author</attribution>
    <simpara>text</text>
</blockquote>
```

visit_moinpage_h(*element*)
There is not really heading in DocBook, but rather section with title. The section is a root tag for all the elements which in the dom tree will be between two heading tags.

So we need to process child manually to determine correctly the children of each section.

A section is closed when we have a new heading with an equal or higher level.

visit_moinpage_line_break(*element*)

visit_moinpage_list(*element*)
Function called to handle the conversion of list.

It will called a specific function to handle (un)ordered list, with the appropriate DocBook tag.

Or a specific function to handle definition list.

visit_moinpage_list_item_body(*element*)

visit_moinpage_note(*element*)

<note note-class="footnote"><note-body>text</note-body></note> –> <footnote><simpara>text</simpara></footnote>

visit_moinpage_object(*element*)
Convert:

```
<object type='image/' xlink:href='uri'/>
```

to:

```
<inlinemediaobject>
      <imageobject>
            <imagedata fileref="uri" />
      </imageobject>
</inlinemediaobject>
```

Similar for video and audio object.

visit_moinpage_p(*element*)
If we have a title attribute for p, we return a para, with a <title> child. Otherwise we return a <simpara>.

visit_moinpage_page(*element*)

visit_moinpage_span(*element*)
The span element is used in the DOM Tree to define some specific formatting. So each attribute will give different resulting tag.

TODO: Add support for text-decoration attribute TODO: Add support for font-size attribute

visit_moinpage_strong(*element*)
 –> <emphasis role=strong>

visit_moinpage_table(*element*)

visit_moinpage_table_body(*element*)

visit_moinpage_table_cell(*element*)

visit_moinpage_table_footer(*element*)

visit_moinpage_table_header(*element*)

visit_moinpage_table_row(*element*)

visit_simple_tag(*element*)

MoinMoin.converter.everything module

MoinMoin - converter for all items (fallback)

Convert any item to a DOM Tree (we just create a link to download it).

class MoinMoin.converter.everything.**Converter**
Bases: object

Convert a unsupported item to DOM Tree.

MoinMoin.converter.highlight module

MoinMoin - Text highlighting converter

class MoinMoin.converter.highlight.**Converter**(*re*)
Bases: object

recurse(*elem*)

MoinMoin.converter.html_in module

MoinMoin - HTML input converter

Converts an XHTML document into an internal document tree.

TODO : Add support for style

class MoinMoin.converter.html_in.**Converter**
Bases: object

Converter html -> .x.moin.document

base_url = ‘’

convert_attributes(*element*)

do_children(*element*)
Function to process the conversion of the child of a given elements.

heading_re = <_sre.SRE_Pattern object>

html_namespace = {<Namespace(‘http://www.w3.org/1999/xhtml’)>: ‘xhtml’}

ignored_tags = set([’frameset’, ‘frame’, ‘select’, ‘noframes’, ‘noscript’, ‘style’, ‘area’, ‘menu’, ‘param’, ‘label’, ‘sc

inline_tags = set([’acronym’, ‘dfn’, ‘abbr’, ‘kbd’, ‘address’])

list_tags = set([’ul’, ‘ol’, ‘dir’])

new(*tag*, *attrib*, *children*)
Return a new element for the DOM Tree

new_copy(*tag*, *element*, *attrib*)
Function to copy one element to the DOM Tree.

It first converts the child of the element, and the element itself.

new_copy_symmetric(*element*, *attrib*)
Create a new QName, with the same tag of the element, but with a different namespace.

Then, we handle the copy normally.

simple_tags = {‘em’: Name(u’emphasis’, u’http://moinmo.in/namespaces/page’), ‘pre’: Name(u’blockcode’, u’http

standard_attributes = set([’style’, ‘class’, ‘alt’, ‘title’])

symmetric_tags = set([’code’, ‘span’, ‘quote’, ‘p’, ‘blockquote’, ‘div’, ‘strong’])

visit(*element*)
Function called at each element, to process it.

It will just determine the namespace of our element, then call a dedicated function to handle conversion for the found namespace.

visit_xhtml(*element*)
Function called to handle the conversion of elements belonging to the XHTML namespace.

We will detect the name of the tag, and apply an appropriate procedure to convert it.

visit_xhtml_a(*element*)
Text –> <a xlink:href=”URI”>Text

visit_xhtml_base(*element*)
Function to store the base url for the relative url of the document

visit_xhtml_big(*element*)
<big>Text</big> –> Text

visit_xhtml_br(*element*)

 –> <line-break />

visit_xhtml_caption(*element*)

visit_xhtml_del(*element*)
~~Text~~ –> ~~Text~~

visit_xhtml_dl(*element*)
Convert a list of definition. The starting structure:

```
<dl>
    <dt>Label 1</dt><dd>Text 1</dd>
    <dt>Label 2</dt><dd>Text 2</dd>
</dl>
```

will be converted to:

```
<list>
    <list-item>
        <list-item-label>Label 1</list-item-label>
        <list-item-body>Text 1</list-item-body>
    </list-item>
    <list-item>
        <list-item-label>Label 2</list-item-label>
        <list-item-body>Text 2</list-item-body>
    </list-item>
</list>
```

visit_xhtml_heading(*element*)
Function to convert an heading tag into the proper element in our moin_page namespace

visit_xhtml_hr(*element*, *min_class=u'moin-hr1'*, *max_class=u'moin-hr6'*, *default_class=u'moin-hr3'*)
<hr /> –> <separator />

visit_xhtml_img(*element*)
 –> <object xlink:href="URI />

visit_xhtml_inline(*element*)
For some specific inline tags (defined in inline_tags) We just return

visit_xhtml_ins(*element*)
<ins>Text</ins> –> <ins>Text</ins>

visit_xhtml_li(*element*)
NB : A list item (<li>) is like the following snippet:

```
<list-item>
    <list-item-label>label</list-item-label>
    <list-item-body>Body</list-item-body>
</list-item>
```

For <li> element, there is no label

visit_xhtml_list(*element*)
Convert a list of items (whatever the type : ordered or unordered) So we have html code like:

```
<ul>
    <li>Item 1</li>
    <li>Item 2</li>
</ul>
```

Which will be converted to:

```
<list>
    <list-item>
        <list-item-body>Item 1</list-item-body>
```

```
    </list-item>
    <list-item>
        <list-item-body>Item 2</list-item-body>
    </list-item>
</list>
```

visit_xhtml_object(*element*)
<object data="href"></object> –> <object xlink="href" />

visit_xhtml_s(*element*)
<s>Text</s> –> <s>Text</s>

visit_xhtml_small(*element*)
<small>Text</small> –> Text

visit_xhtml_strike(*element*)
<strike>Text</strike> –> <s>Text</s> # strike is not a valid tag in html5

visit_xhtml_sub(*element*)
_{Text} –> Text

visit_xhtml_sup(*element*)
^{Text} –> Text

visit_xhtml_table(*element*)

visit_xhtml_tbody(*element*)

visit_xhtml_td(*element*, *attr={}*)

visit_xhtml_tfoot(*element*)

visit_xhtml_th(*element*)
html_out does not support th tags, so we do td tags like all other converters.

visit_xhtml_thead(*element*)

visit_xhtml_tr(*element*)

visit_xhtml_u(*element*)
<u>Text</u> –> <u>Text</u>

MoinMoin.converter.html_out module

MoinMoin - HTML output converter

Converts an internal document tree into a HTML tree.

class MoinMoin.converter.html_out.**Attribute**(*key*)
Bases: object

Adds the attribute with the HTML namespace to the output.

key

class MoinMoin.converter.html_out.**Attributes**(*element*)
Bases: object

convert()

get(*name*)

namespaces_valid_output = frozenset([<Namespace('http://www.w3.org/1999/xhtml')>])

visit_class = <MoinMoin.converter.html_out.Attribute object>

visit_id = <MoinMoin.converter.html_out.Attribute object>

visit_number_columns_spanned = <MoinMoin.converter.html_out.Attribute object>

visit_number_rows_spanned = <MoinMoin.converter.html_out.Attribute object>

visit_style = <MoinMoin.converter.html_out.Attribute object>

visit_title = <MoinMoin.converter.html_out.Attribute object>

visit_type = <MoinMoin.converter.html_out.Attribute object>

class MoinMoin.converter.html_out.**Converter**
Bases: object

Converter application/x.moin.document -> application/x.moin.document

direct_inline_tags = set(['dfn', 'abbr', 'kbd', 'address'])

do_children(*element*)

eval_object_type(*mimetype*, *href*)
Returns the type of an object as a str, one of the following: img, video, audio, object

namespaces_visit = {<Namespace('http://moinmo.in/namespaces/page')>: 'moinpage'}

new_copy(*tag*, *element*, *attrib={}*)

visit(*elem*)

visit_moinpage(*elem*)

visit_moinpage_a(*elem*, *_tag_html_a=Name(u'a', u'http://www.w3.org/1999/xhtml')*, *_tag_html_href=Name(u'href', u'http://www.w3.org/1999/xhtml')*, *_tag_xlink_href=Name(u'href', u'http://www.w3.org/1999/xlink')*)

visit_moinpage_admonition(*elem*)
Used by ReST and docbook.

visit_moinpage_block_comment(*elem*)

visit_moinpage_blockcode(*elem*)

visit_moinpage_blockquote(*elem*)

visit_moinpage_code(*elem*)

visit_moinpage_del(*elem*)

visit_moinpage_div(*elem*)

visit_moinpage_emphasis(*elem*)

visit_moinpage_h(*elem*)

visit_moinpage_inline_part(*elem*)

visit_moinpage_ins(*elem*)

visit_moinpage_line_break(*elem*)

visit_moinpage_list(*elem*)

visit_moinpage_list_item(*elem*)
Used for markdown definition lists.

Compared to moinwiki and ReST parsers, the markdown parser creates definition lists using only one list-item tag.name for entire list where moinwiki and ReST have one list-item tag.name for each entry in list.

visit_moinpage_nowiki(*elem*)

visit_moinpage_object(*elem*)
elem of type img are converted to img tags here, others are left as object tags.

We do not use Attributes.convert to convert all attributes, but copy selected attributes and follow html5 validation rules to place right attributes within img and object tags.

visit_moinpage_p(*elem*)

visit_moinpage_page(*elem*)

visit_moinpage_part(*elem*)

visit_moinpage_quote(*elem*)

visit_moinpage_s(*elem*)

visit_moinpage_samp(*elem*)

visit_moinpage_separator(*elem*)

visit_moinpage_span(*elem*)

visit_moinpage_strong(*elem*)

visit_moinpage_table(*elem*)

visit_moinpage_table_cell(*elem*)

visit_moinpage_table_row(*elem*)

visit_moinpage_u(*elem*)

class MoinMoin.converter.html_out.**ConverterDocument**
Bases: *MoinMoin.converter.html_out.ConverterPage*

Converter application/x.moin.document -> application/xhtml+xml

class MoinMoin.converter.html_out.**ConverterPage**
Bases: *MoinMoin.converter.html_out.Converter*

Converter application/x.moin.document -> application/x-xhtml-moin-page

create_footnotes(*top*)
Return footnotes formatted into an ET structure.

visit(*elem*, *_tag_moin_page_page_href=Name(u'page-href', u'http://moinmo.in/namespaces/page')*)

visit_moinpage_h(*elem*, *_tag_html_id=Name(u'id', u'http://www.w3.org/1999/xhtml')*)

visit_moinpage_note(*elem*)

visit_moinpage_table_of_content(*elem*)

exception MoinMoin.converter.html_out.**ElementException**
Bases: exceptions.RuntimeError

class MoinMoin.converter.html_out.**SpecialId**
Bases: object

gen_id(*id*)

gen_text(*text*)

get_id(*id*)

zero_id(*id*)

class MoinMoin.converter.html_out.**SpecialPage**
Bases: object

add_footnote(*elem*)

add_heading(*elem*, *level*, *id=None*)

add_toc(*elem*, *maxlevel*)

extend(*page*)

footnotes()

headings(*maxlevel*)

remove_footnotes()

tocs()

MoinMoin.converter.html_out.**convert_getlink_to_showlink**(*href*)
If the incoming transclusion reference is within this domain, then remove "+get/<revision number>/".

MoinMoin.converter.html_out.**mark_item_as_transclusion**(*elem*, *href*)
Return elem after adding a "moin-transclusion" class and a "data-href" attribute with a link to the transcluded item.

On the client side, a Javascript function will wrap the element (or a parent element) in a span or div and 2 overlay siblings will be created.

MoinMoin.converter.html_out.**style_attr_filter**(*style*)
If allow_style_attributes option is True check style attribute for suspect strings, else return ''.

MoinMoin.converter.image_in module

MoinMoin - Image converter

Convert image to <object> tag for the DOM Tree.

class MoinMoin.converter.image_in.**Converter**(*input_type*)
Bases: object

Convert an image to the corresponding <object> in the DOM Tree

MoinMoin.converter.include module

MoinMoin - Include handling

Expands include elements in an internal Moin document.

Although this module is named include.py, many comments within and the moin docs use the word transclude as defined by http://www.linfo.org/transclusion.html, etc.

Adjusting the DOM

After expanding the include elements, in many cases it is necessary to adjust the DOM to prevent the generation of invalid HTML. Using a simple example, " {{SomeItem}} ", the starting DOM structure created by the moinwiki_in.py (or other parser) is:

```
Page > Body > P > Include
```

After expansion of the Include, the structure will be:

```
Page > Body > P > Page > Body > (P | Div | Object |...)
```

moinwiki_in.py (or other parser) does not adjust the DOM structure based upon whether the contents of the transcluded item are inline or block. Sometime after include processing is complete, html_out.py will convert the transcluded Body > Page into a Div or Span wrapping the transclusion contents.

This works well for things like " ||mytable||{{BlockOrInline}}|| " where almost any type of element is valid within a table cell's td.

But without DOM adjustment, " {{Block}} " will generate invalid HTML because html_out.py will convert the DOM structure:

```
Page > Body > P > Page > Body > (Pre | Div | P, P... | ...)
```

into:

```
...<body><p><div>...</div></p></body>...
```

where the </p> is invalid.

In some cases it is desirable to coerce a transcluded small image or phrase into a inline element embedded within a paragraph. Here html_out.py will wrap the transclusion in a Span rather than a Div or convert a P-tag containing a phrase into a Span:

```
"My pet {{bird.jpg}} flys.", "[[SomePage|{{Logo.png}}]]" or "Yes, we have {{no}}␣
↪bananas."
```

In complex cases where a block level item is transcluded within the midst of several levels of text markup, such as:

```
"plain ''italic '''bold {{BlockItem}} bold''' italic'' plain"
```

then we must avoid generating invalid html like:

```
<p>plain <emphasis>italic <strong>bold <div>
...</div> bold</strong> italic</emphasis> plain</p>
```

where ... contains the transcluded item, but rather:

```
<p>plain <emphasis>italic <strong>bold</strong></emphasis></p><div>
...</div><p><emphasis><strong> bold</strong> italic</emphasis> plain</p>
```

In these complex cases, we must build a DOM structure that will replace the containing element's parent, grand-parent, great-grand-parent...

When a block element is embedded within a comment, it is important that the class="comment" is copied to the transclusion to provide the show/hide and highlighted styles normally applied to comments:

```
/* normal ''italic ~-small {{detail.csv}} small-~ italic'' normal */
```

Conveniently, the class="comment" is added to the span element within the moinwiki_in.py parser and is available to include.py. However, the moin-big and moin-small classes are applied to span elements by html_out.py so those classes are not available. Italic, bold, stroke, and underline styling effects are implemented through specialized tags rather than CSS classes. In the example above, only class="comment" will be applied to detail.csv.

class MoinMoin.converter.include.**Converter**

Bases: object

recurse(*elem*, *page_href*)

tag_a = Name(u'a', u'http://moinmo.in/namespaces/page')

tag_div = Name(u'div', u'http://moinmo.in/namespaces/page')

tag_h = Name(u'h', u'http://moinmo.in/namespaces/page')

tag_href = Name(u'href', u'http://www.w3.org/1999/xlink')

tag_outline_level = Name(u'outline-level', u'http://moinmo.in/namespaces/page')

tag_page_href = Name(u'page-href', u'http://moinmo.in/namespaces/page')

tag_xi_href = Name(u'href', u'http://www.w3.org/2001/XInclude')

tag_xi_include = Name(u'include', u'http://www.w3.org/2001/XInclude')

tag_xi_xpointer = Name(u'xpointer', u'http://www.w3.org/2001/XInclude')

class MoinMoin.converter.include.**XPointer**(*input*)

Bases: list

Simple XPointer parser

class **Entry** (*name*, *data*)
Bases: object

data

data_unescape

name

XPointer.**tokenizer_re** = <_sre.SRE_Pattern object>

XPointer.**tokenizer_rules** = '\n # Match escaped syntax elements\n \\^[()^]\n |\n (?P<bracket_open> \\( )\n |\n

MoinMoin.converter.link module

MoinMoin - Link converter

Expands all links in an internal Moin document, including interwiki and special wiki links.

class MoinMoin.converter.link.**ConverterBase**
Bases: object

absolute_path (*path*, *current_page_path*)
Converts a relative iri path into an absolute one

Parameters

- **path** (Iri.path) – the relative path to be converted
- **current_page_path** (Iri.path) – the path of the page where the link is

Returns the absolute equivalent of the relative path

Return type *Iri.path*

handle_external_links (*elem*, *link*)

handle_wiki_links (*elem*, *link*)

handle_wiki_transclusions (*elem*, *link*)

handle_wikilocal_links (*elem*, *link*, *page_name*)

handle_wikilocal_transclusions (*elem*, *link*, *page_name*)

traverse_tree (*elem*, *page=None*, *_ConverterBase__tag_page_href=Name(u'page-href'*, *u'http://moinmo.in/namespaces/page')*, *_ConverterBase__tag_link=Name(u'href'*, *u'http://www.w3.org/1999/xlink')*, *_ConverterBase__tag_include=Name(u'href'*, *u'http://www.w3.org/2001/XInclude')*)
Traverses the tree and handles each element appropriately

class MoinMoin.converter.link.**ConverterExternOutput**
Bases: *MoinMoin.converter.link.ConverterBase*

handle_external_links (*elem*, *input*)

handle_wiki_links (*elem*, *input*)

handle_wikilocal_links (*elem*, *input*, *page*)

class MoinMoin.converter.link.**ConverterItemRefs** (***kw*)
Bases: *MoinMoin.converter.link.ConverterBase*

determine all links and transclusions to other wiki items in this document

get_external_links ()
return a list of unicode external links target item names

get_links ()
return a list of unicode link target item names

get_transclusions()
Return a list of unicode transclusion item names.

handle_external_links(*elem*, *input*)
Adds the link item from the input param to self.external_links :param elem: the element of the link :param input: the iri of the link

handle_wikilocal_links(*elem*, *input*, *page*)
Adds the link item from the input param to self.links :param elem: the element of the link :param input: the iri of the link :param page: the iri of the page where the link is

handle_wikilocal_transclusions(*elem*, *input*, *page*)
Adds the transclusion item from input argument to self.transclusions :param elem: the element of the transclusion :param input: the iri of the transclusion :param page: the iri of the page where the transclusion is

MoinMoin.converter.macro module

MoinMoin - Macro handling

Expands all macro elements in an internal Moin document.

class MoinMoin.converter.macro.**Converter**
Bases: object

handle_macro(*elem*, *page*)

recurse(*elem*, *page*)

MoinMoin.converter.markdown_in module

MoinMoin - Markdown input converter

http://daringfireball.net/projects/markdown/

class MoinMoin.converter.markdown_in.**Converter**
Bases: object

convert_align_to_class(*attrib*)

convert_attributes(*element*)

convert_embedded_markup(*node*)
Recurse through tree looking for embedded markup.

Parameters **node** – a tree node

convert_invalid_p_nodes(*node*)
Processing embedded HTML tags within markup or output from extensions with embedded markup can result in invalid HTML output caused by <p> tags enclosing a block element.

The solution is to search for these occurances and change the <p> tag to a .

Parameters **node** – a tree node

count_lines(*text*)
Create a list of line numbers corresponding to the first line of each markdown block.

The markdown parser does not provide text line numbers nor is there an easy way to add line numbers. As an alternative, we try to split the input text into the same blocks as the parser does, then calculate the starting line number of each block. The list will be processed by the do_children method above.

This method has unresolved problems caused by splitting the text into blocks based upon the presence of 2 adjacent line end characters, including:

•blank lines within lists create separate blocks

- omitting a blank line after a heading combines 2 elements into one block
- using more than one blank lines between blocks

The net result is we either have too few or too many line numbers in the generated list which will cause the double-click-to-edit autoscroll textarea to sometimes be off by several lines.

do_children (*element*, *add_lineno=False*)

embedded_markup (*text*)
Per http://meta.stackexchange.com/questions/1777/what-html-tags-are-allowed-on-stack-exchange-sites markdown markup allows users to specify several "safe" HTML tags within a document. These tags include:

a b blockquote code del dd dl dt em h1 h2 h3 i img kbd li ol p pre s sup sub strong strike ul br hr

In addition, some markdown extensions output raw HTML tags (e.g. fenced outputs "<pre><code>..."). To prevent the <, > characters from being escaped, the embedded tags are converted to nodes by using the converter in html_in.py.

heading_re = <_sre.SRE_Pattern object>

ignored_tags = set(['frameset', 'frame', 'select', 'noframes', 'noscript', 'style', 'area', 'menu', 'param', 'label', 'sc

inline_tags = set(['acronym', 'dfn', 'abbr', 'kbd', 'address'])

list_tags = set(['ul', 'ol'])

new (*tag*, *attrib*, *children*)
Return a new element for the DOM Tree

new_copy (*tag*, *element*, *attrib*)
Function to copy one element to the DOM Tree.

It first converts the child of the element, and the element itself.

new_copy_symmetric (*element*, *attrib*)
Create a new QName, with the same tag of the element, but with a different namespace.

Then, we handle the copy normally.

simple_tags = {'em': Name(u'emphasis', u'http://moinmo.in/namespaces/page'), 'pre': Name(u'blockcode', u'http

standard_attributes = set(['style', 'class', 'title'])

symmetric_tags = set(['code', 'quote', 'p', 'blockquote', 'div', 'strong'])

visit (*element*)

visit_a (*element*)

visit_big (*element*)

visit_br (*element*)

visit_del (*element*)

visit_heading (*element*)
Function to convert an heading tag into a proper element in our moin_page namespace

visit_hr (*element*, *default_class=u'moin-hr3'*)

visit_img (*element*)
 –> <object xlink:href="URI />

visit_inline (*element*)
For some specific inline tags (defined in inline_tags) We just return

visit_ins (*element*)

visit_li(*element*)
NB : A list item (<li>) is like the following snippet:

```
<list-item>
    <list-item-label>label</list-item-label>
    <list-item-body>Body</list-item-body>
</list-item>
```

For <li> element, there is no label

visit_list(*element*)
Convert a list of item (whatever the type : ordered or unordered) So we have html code like:

```
<ul>
    <li>Item 1</li>
    <li>Item 2</li>
</ul>
```

Which will be converted to:

```
<list>
    <list-item>
        <list-item-body>Item 1</list-item-body>
    </list-item>
    <list-item>
        <list-item-body>Item 2</list-item-body>
    </list-item>
</list>
```

visit_object(*element*)
<object data="href"></object> –> <object xlink="href" />

visit_s(*element*)

visit_small(*element*)

visit_strike(*element*)

visit_sub(*element*)

visit_sup(*element*)

visit_td(*element*)

visit_th(*element*)

visit_u(*element*)

MoinMoin.converter.markdown_in.**postproc_text**(*markdown*, *text*)
Removes HTML or XML character references and entities from a text string.

Parameters **text** – The HTML (or XML) source text.

Returns The plain text, as a Unicode string, if necessary.

MoinMoin.converter.mediawiki_in module

MoinMoin - Media Wiki input converter

class MoinMoin.converter.mediawiki_in.**Converter**
Bases: MoinMoin.converter._wiki_macro.ConverterMacro

class **Mediawiki_preprocessor**
Bases: object

class **Preprocessor_tag**(*name=''*, *text=''*, *tag=''*, *status=True*)
Bases: object

Converter.Mediawiki_preprocessor.**all_tags = ['br', 'blockquotedel', 'pre', 'code', 'tt', 'nowiki', 'r**

Converter.Mediawiki_preprocessor.**block_tags = ['blockquote']**

Converter.Mediawiki_preprocessor.**nowiki_tags = ['pre', 'code', 'tt', 'nowiki']**

Converter.Mediawiki_preprocessor.**pop**()

Converter.Mediawiki_preprocessor.**push**(*status=[]*)

Converter.**block = ('(?P<line> ^ \\s* $)', '\n ^\n (?P<table>\n \\{\\|\n \\s*\n (?P<table_args> .*?)\n)\n $\n ', '\n (?P**

Converter.**block_head = '\n (?P<head>\n ^\n \\s*\n (?P<head_head> =+)\n \\s*\n (?P<head_text> .*?)\n \\s*\n (**

Converter.**block_head_repl**(*_iter_content*, *stack*, *head*, *head_head*, *head_text*)

Converter.**block_line = '(?P<line> ^ \\s* $)'**

Converter.**block_line_repl**(*_iter_content*, *stack*, *line*)

Converter.**block_re = <_sre.SRE_Pattern object at 0x2cbf360>**

Converter.**block_separator = '(?P<separator> ^ \\s* -{4,} \\s* $)'**

Converter.**block_separator_repl**(*_iter_content*, *stack*, *separator*)

Converter.**block_table = '\n ^\n (?P<table>\n \\{\\|\n \\s*\n (?P<table_args> .*?)\n)\n $\n '**

Converter.**block_table_lines**(*iter_content*)
Unescaping generator for the lines in a table block

Converter.**block_table_repl**(*iter_content*, *stack*, *table*, *table_args=''*)

Converter.**block_text = '(?P<text> .+)'**

Converter.**block_text_repl**(*_iter_content*, *stack*, *text*)

classmethod Converter.**factory**(*input*, *output*, ***kw*)

Converter.**indent = '\n ^\n (?P<indent> [*#:]*)\n (?P<list_begin>\n (?P<list_definition> ;\n)\n \\s*\n |\n (?P<list_**

Converter.**indent_iter**(*iter_content*, *line*, *level*, *is_list*)

Converter.**indent_re = <_sre.SRE_Pattern object at 0x2c64980>**

Converter.**indent_repl**(*iter_content*, *stack*, *line*, *indent*, *text*, *list_begin=None*, *list_definition=None*, *list_definition_text=None*, *list_numbers=None*, *list_bullet=None*, *list_none=None*)

Converter.**inline = ('\n (?P<link>\n \\[\\[\n \\s*\n (\n (?P<link_url>\n [a-zA-Z0-9+.-]+\n :\n [^|]+?\n)\n |\n (?P<li**

Converter.**inline_blockquote = '\n (?P<blockquote>\n (?P<blockquote_begin>\n \\<blockquote.*?\\>\n)\n |\n**

Converter.**inline_blockquote_repl**(*stack*, *blockquote*, *blockquote_begin=None*, *blockquote_end=None*)

Converter.**inline_breakline = '\n (?P<breakline>\n \\<br\\ \\/\\>\n)\n '**

Converter.**inline_breakline_repl**(*stack*, *breakline*)

Converter.**inline_comment = '\n (?P<comment>\n (?P<comment_begin>\n (^|(?<=\\s))\n /*\n \\s+\n)\n |\n (?P**

Converter.**inline_comment_repl**(*stack*, *comment*, *comment_begin=None*, *comment_end=None*)

Converter.**inline_delete = '\n (?P<delete>\n \\<del\\>\n |\n \\<\\/del\\>\n)\n '**

Converter.**inline_delete_repl**(*stack*, *delete*)

Converter.**inline_emphstrong = "\n (?P<emphstrong>\n '{2,6}\n (?=\n [^']+\n (?P<emphstrong_follow>\n '{2,**

Converter.**inline_emphstrong_repl**(*stack*, *emphstrong*, *emphstrong_follow=''*)

Converter.**inline_entity = '\n (?P<entity>\n &\n (?:\n # symbolic entity, like ü\n [0-9a-zA-Z]{2,6}\n |\n**

Converter.**inline_entity_repl**(*stack*, *entity*)

Converter.**inline_footnote** = '\n (?P<footnote>\n (?P<footnote_begin>\n \\<ref.*?\\>\n)\n |\n (?P<footnote_en

Converter.**inline_footnote_repl**(*stack*, *footnote*, *footnote_begin=None*, *footnote_text=None*, *footnote_end=None*, *footnote_start=None*)

Converter.**inline_insert** = '\n (?P<insert>\n \\<ins\\>\n |\n \\<\\/ins\\>\n)\n '

Converter.**inline_insert_repl**(*stack*, *insert*)

Converter.**inline_link** = '\n (?P<link>\n \\[\\[\n \\s*\n (\n (?P<link_url>\n [a-zA-Z0-9+.-]+\n :\n [^|]+?\n)\n |\n (

Converter.**inline_link_repl**(*stack*, *link*, *link_url=None*, *link_item=None*, *link_args=u''*, *external_link_url=None*, *alt_text=u''*)

Handle all kinds of links.

Converter.**inline_nowiki** = '\n (?P<nowiki>\n <nowiki>\n (?P<nowiki_text> .*?)\n </nowiki>\n |\n <pre>\n (?

Converter.**inline_nowiki_repl**(*stack*, *nowiki*, *nowiki_text=None*, *nowiki_text_pre=None*, *pre_args=''*, *nowiki_text_code=None*, *nowiki_text_tt=None*)

Converter.**inline_re** = <_sre.SRE_Pattern object at 0x2cc3e70>

Converter.**inline_strike** = '\n (?P<strike>\n \\<s\\>\n |\n \\<\\/s\\>\n)\n '

Converter.**inline_strike_repl**(*stack*, *strike*)

Converter.**inline_subscript** = '\n (?P<subscript>\n _{\n (?P<subscript_text> .*?)\n}\n)\n '

Converter.**inline_subscript_repl**(*stack*, *subscript*, *subscript_text*)

Converter.**inline_superscript** = '\n (?P<superscript>\n ^{\n (?P<superscript_text> .*?)\n}\n)\n '

Converter.**inline_superscript_repl**(*stack*, *superscript*, *superscript_text*)

Converter.**inline_underline** = '\n (?P<underline>\n \\<u\\>\n |\n \\<\\/u\\>\n)\n '

Converter.**inline_underline_repl**(*stack*, *underline*)

Converter.**inlinedesc** = ('\n (?P<breakline>\n \\<br\\ \\/\\>\n)\n ', '\n (?P<nowiki>\n <nowiki>\n (?P<nowiki_te

Converter.**inlinedesc_re** = <_sre.SRE_Pattern object at 0x2cc16a0>

Converter.**parse_args**(*input*)

Parses media wiki arguments, this is taken from _args_wiki > parse function. The primary difference being that mediawiki breaks on pipes whereas the default parser breaks on spaces. Apart from that this parser also supports a few extra characters such as "<, >, ., /", mostly for URL linking.

Parameters **input** – can be like a|b|c=f|something else caption|g='long caption'|link=http://google.com

Returns Arguments instance

Converter.**parse_block**(*iter_content*, *arguments*)

Converter.**parse_inline**(*text*, *stack*, *inline_re*)

Recognize inline elements within the given text

Converter.**table** = '\n ^\n (?P<table>\n \\{\\|\n \\s*\n (?P<table_args> .*?)\n)\n $\n '

Converter.**table_end** = '\n ^\n (?P<table_end>\n \\|\\}\n \\s*\n)\n $\n '

Converter.**table_end_re** = <_sre.SRE_Pattern object>

Converter.**table_re** = <_sre.SRE_Pattern object>

Converter.**tablerow** = '\n ^\n [|!]\n (?P<tablerow>\n (?P<caption> \\+.*)\n |\n (?P<newrow> \\-\\s*)\n |\n (?P<cel

Converter.**tablerow_re** = <_sre.SRE_Pattern object>

MoinMoin.converter.moinwiki19_in module

MoinMoin - Moin Wiki 1.9 input converter

class MoinMoin.converter.moinwiki19_in.**ConverterFormat19**
Bases: *MoinMoin.converter.moinwiki_in.Converter*

classmethod factory (*input*, *output*, ***kw*)

inline = (‘\n (?P<link>\n \\[\\[\n \\s*\n (\n (?P<link_url>\n (http|https|ftp|file|mailto|nntp|news|ssh|telnet|irc|ircs|xmp

inline_freelink = u’\n (?:\n (?<![ABCDEFGHIJKLMNOPQRSTUVWXYZ\xc0\xc1\xc2\xc3\xc4\xc5\xc6\xc7\xc8

inline_freelink_repl (*stack*, *freelink*, *freelink_bang=None*, *freelink_interwiki_page=None*, *freelink_interwiki_ref=None*, *freelink_page=None*, *freelink_email=None*)

inline_re = <_sre.SRE_Pattern object at 0x45e0ac0>

inline_url = ‘\n (?P<url>\n (\n ^\n |\n (?<=\n \\s\n |\n [.,:;!?()/=]\n)\n)\n (?P<url_target>\n (http|https|ftp|file|mailt

inline_url_repl (*stack*, *url*, *url_target*)

MoinMoin.converter.moinwiki_in module

MoinMoin - Moin Wiki input converter

class MoinMoin.converter.moinwiki_in.**Converter**
Bases: MoinMoin.converter._wiki_macro.ConverterMacro

block = (‘(?P<line> ^ \\s* $)’, ‘\n (?P<comment>\n ^ \\#\\#.*$\n)\n ‘, ‘\n (?P<head>\n ^\n \\s*\n (?P<head_head> =+

block_comment = ‘\n (?P<comment>\n ^ \\#\\#.*$\n)\n ‘

block_comment_repl (*_iter_content*, *stack*, *comment*)

block_head = ‘\n (?P<head>\n ^\n \\s*\n (?P<head_head> =+)\n \\s+ # for better moin 1.x compatibility, we require

block_head_repl (*_iter_content*, *stack*, *head*, *head_head*, *head_text*)

block_line = ‘(?P<line> ^ \\s* $)’

block_line_repl (*_iter_content*, *stack*, *line*)

block_macro = ‘\n ^\n \\s*\n (?P<macro>\n <<\n (?P<macro_name> \\w+)\n (\n \\(\n (?!.*>>.*>>)\n (?P<macro_arg

block_macro_repl (*_iter_content*, *stack*, *macro*, *macro_name*, *macro_args=u’‘*)
Handles macros using the placeholder syntax.

Arguments are passed as a single positional parameter, each macro must parse as required.

block_nowiki = ‘\n (?P<nowiki>\n ^\n \\s*\n (?P<nowiki_marker> \\{{3,})\n \\s* # spaces not defined here, but acc

block_nowiki_lines (*iter_content*, *marker_len*)
Unescaping generator for the lines in a nowiki block

block_nowiki_repl (*iter_content*, *stack*, *nowiki*, *nowiki_marker*, *nowiki_interpret=’‘*, *nowiki_name=None*, *nowiki_args=None*, *optional_args=None*)

block_re = <_sre.SRE_Pattern object at 0x45ab5b0>

block_separator = ‘(?P<separator> ^ \\s* -{4,} \\s* $)’

block_separator_repl (*_iter_content*, *stack*, *separator*, *hr_class=u’moin-hr{0}’*)

block_table = ‘\n ^\n \\s*\n (?P<table>\n \\|\\|\n .*\n)\n \\|\\|\n \\s*\n $\n ‘

block_table_repl (*iter_content*, *stack*, *table*)

block_table_row (*content*, *stack*, *table*)

block_text = ‘(?P<text> .+)’

block_text_repl(*_iter_content*, *stack*, *text*)

classmethod factory(*input*, *output*, ***kw*)

header_footer_re = <_sre.SRE_Pattern object>

header_footer_separator = '\n ^\n \\s*\n (?P<table_sep>\n ===+\n)\n \\s*\n $\n '

indent = '\n ^\n (?P<indent> \\s*)\n (?P<list_begin>\n (?P<list_definition>\n (?P<list_definition_text> .*?)\n ::\n)

indent_iter(*iter_content*, *line*, *level*)

indent_re = <_sre.SRE_Pattern object at 0x45ab0b0>

indent_repl(*iter_content*, *stack*, *line*, *indent*, *text*, *list_begin=None*, *list_definition=None*, *list_definition_text=None*, *list_numbers=None*, *list_alpha=None*, *list_roman=None*, *list_bullet=None*, *list_start_number=None*, *list_start_roman=None*, *list_start_alpha=None*, *list_none=None*)

inline = ('\n (?P<link>\n \\[\\[\n \\s*\n (\n (?P<link_url>\n (http|https|ftp|file|mailto|nntp|news|ssh|telnet|irc|ircs|xmp

inline_comment = '\n (?P<comment>\n (?P<comment_begin>\n (^|(?<=\\s))\n /*\n \\s+\n)\n |\n (?P<comment_en

inline_comment_repl(*stack*, *comment*, *comment_begin=None*, *comment_end=None*)

inline_emphstrong = "\n (?P<emphstrong>\n '{2,6}\n (?=\n [^']+\n (?P<emphstrong_follow>\n '{2,3}\n (?!')\n)\n

inline_emphstrong_repl(*stack*, *emphstrong*, *emphstrong_follow=''*)

inline_entity = '\n (?P<entity>\n &\n (?:\n # symbolic entity, like ü\n [0-9a-zA-Z]{2,6}\n |\n # numeric deci

inline_entity_repl(*stack*, *entity*)

inline_link = '\n (?P<link>\n \\[\\[\n \\s*\n (\n (?P<link_url>\n (http|https|ftp|file|mailto|nntp|news|ssh|telnet|irc|irc

inline_link_repl(*stack*, *link*, *link_url=None*, *link_item=None*, *link_text=None*, *link_args=None*, *link_interwiki_site=None*, *link_interwiki_item=None*)

Handle all kinds of links.

inline_macro = '\n (?P<macro>\n <<\n (?P<macro_name> \\w+)\n (\n \\(\n (?P<macro_args> .*?)\n \\)\n)?\n \\s*

inline_macro_repl(*stack*, *macro*, *macro_name*, *macro_args=u''*)

Handles macros using the placeholder syntax.

inline_nowiki = '\n (?P<nowiki>\n {{{\n (?P<nowiki_text>.*?}*)\n }}}\n |\n '\n (?P<nowiki_text_backtick> .*?)\n

inline_nowiki_repl(*stack*, *nowiki*, *nowiki_text=None*, *nowiki_text_backtick=None*)

inline_object = '\n (?P<object>\n {{\n \\s*\n (\n (?P<object_url>\n [a-zA-Z0-9+.-]+\n ://\n [^|]+?\n)\n |\n (?P<obj

inline_object_repl(*stack*, *object*, *object_url=None*, *object_item=None*, *object_text=None*, *object_args=None*)

Handles objects included in the page.

inline_re = <_sre.SRE_Pattern object at 0x45af9e0>

inline_size = '\n (?P<size>\n (?P<size_begin>\n ~[-+]\n)\n |\n (?P<size_end>\n [-+]~\n)\n)\n '

inline_size_repl(*stack*, *size*, *size_begin=None*, *size_end=None*)

inline_strike = '\n (?P<strike>\n (?P<strike_begin>)\n –\\(\n |\n \\)–\n)\n '

inline_strike_repl(*stack*, *strike*, *strike_begin=None*)

inline_subscript = '\n (?P<subscript>\n ,,\n (?P<subscript_text> .*?)\n ,,\n)\n '

inline_subscript_repl(*stack*, *subscript*, *subscript_text*)

inline_superscript = '\n (?P<superscript>\n \\^\n (?P<superscript_text> .*?)\n \\^\n)\n '

inline_superscript_repl(*stack*, *superscript*, *superscript_text*)

inline_underline = '\n (?P<underline>\n __\n)\n '

inline_underline_repl(*stack*, *underline*)

inlinedesc = (‘\n (?P<macro>\n <<\n (?P<macro_name> \\w+)\n (\n \\(\n (?P<macro_args> .*?)\n \\)\n)?\n \\s*\n

inlinedesc_re = <_sre.SRE_Pattern object at 0x45aec80>

macro_text (*text*)
Return an ET tree branch representing the markup present in the input text. Used for FootNotes, etc.

nowiki_end = ‘\n ^\n \\s*\n (?P<marker> }{3,})\n \\s*\n $\n ‘

nowiki_end_re = <_sre.SRE_Pattern object>

parse_block (*iter_content*, *arguments*)

parse_inline (*text*, *stack*, *inline_re*)
Recognize inline elements within the given text

table = ‘\n ^\n \\s*\n (?P<table>\n \\|\\|\n .*\n)\n \\|\\|\n \\s*\n $\n ‘

table_re = <_sre.SRE_Pattern object>

tablerow = ‘\n (?P<cell>\n (?P<cell_marker>\n (\\|\\|)+\n)\n (\n <\n (?P<cell_args> ([^<])*?)\n >\n)?\n (?P<cell_tex

tablerow_cell_repl (*stack*, *table*, *row*, *cell*, *cell_marker*, *cell_text*, *cell_args=None*)

tablerow_re = <_sre.SRE_Pattern object>

MoinMoin.converter.moinwiki_out module

MoinMoin - Moinwiki markup output converter

Converts an internal document tree into moinwiki markup.

class MoinMoin.converter.moinwiki_out.**Converter**
Bases: object

Converter application/x.moin.document -> text/x.moin.wiki

classmethod factory (*input*, *output*, ***kw*)

namespaces = {<Namespace(‘http://www.w3.org/2001/XInclude’)>: ‘xinclude’, <Namespace(‘http://moinmo.in/nam

open (*elem*)

open_children (*elem*)

open_moinpage (*elem*)

open_moinpage_a (*elem*)

open_moinpage_block_comment (*elem*)

open_moinpage_blockcode (*elem*)

open_moinpage_body (*elem*)

open_moinpage_code (*elem*)

open_moinpage_del (*elem*)

open_moinpage_div (*elem*)

open_moinpage_emphasis (*elem*)

open_moinpage_h (*elem*)

open_moinpage_inline_part (*elem*)

open_moinpage_ins (*elem*)

open_moinpage_line_break (*elem*)

open_moinpage_list (*elem*)

open_moinpage_list_item (*elem*)

open_moinpage_list_item_body(*elem*)

open_moinpage_list_item_label(*elem*)

open_moinpage_note(*elem*)

open_moinpage_nowiki(*elem*)
{{{#!wiki ... or {{{#!highlight ... etc.

open_moinpage_object(*elem*)
Process objects and xincludes.

open_moinpage_p(*elem*)

open_moinpage_page(*elem*)

open_moinpage_part(*elem*)

open_moinpage_samp(*elem*)

open_moinpage_separator(*elem*, *hr_class_prefix=u'moin-hr'*)

open_moinpage_span(*elem*)

open_moinpage_strong(*elem*)

open_moinpage_table(*elem*)

open_moinpage_table_body(*elem*)

open_moinpage_table_cell(*elem*)

open_moinpage_table_footer(*elem*)

open_moinpage_table_header(*elem*)

open_moinpage_table_of_content(*elem*)

open_moinpage_table_row(*elem*)

open_xinclude(*elem*)
Processing of transclusions is similar to objects.

supported_tag = {'xinclude': ('include',), 'moinpage': ('a', 'blockcode', 'break_line', 'code', 'div', 'emphasis', 'h',

class MoinMoin.converter.moinwiki_out.**Moinwiki**
Bases: object

Moinwiki syntax elements It's dummy

a_close = u']]'

a_open = u'[['

a_separator = u'|'

definition_list_marker = u'::'

emphasis = u"'"

h = u'='

larger_close = u'+~'

larger_open = u'~+'

**linebreak = u'<
>'**

list_type = {(u'ordered', None): u'1.', (u'unordered', u'no-bullet'): u'.', (u'ordered', u'upper-alpha'): u'A.', (u'un

monospace = u'`'

object_close = u'}}'

object_open = u'{{'

p = u'\n'

samp_close = u'}}}'

samp_open = u'{{{'

separator = u'—-'

smaller_close = u'-~'

smaller_open = u'~-'

stroke_close = u')–'

stroke_open = u'–('

strong = u'"'"'

table_marker = u'||'

underline = u'__'

verbatim_close = u'}'

verbatim_open = u'{'

MoinMoin.converter.nonexistent_in module

MoinMoin - converter for non-existing items

Convert a non-existent item to the DOM Tree.

class MoinMoin.converter.nonexistent_in.**Converter**
Bases: object

Convert a non-existing item to DOM Tree.

MoinMoin.converter.opendocument_in module

MoinMoin - OpenDocument Format (ODF) input converter

ODF documents can be created with OpenOffice.org, Libre Office and other software.

class MoinMoin.converter.opendocument_in.**OpenDocumentIndexingConverter**
Bases: object

MoinMoin.converter.opendocument_in.**OpenOfficeIndexingConverter**
alias of *OpenDocumentIndexingConverter*

MoinMoin.converter.pdf_in module

MoinMoin - PDF input converter

class MoinMoin.converter.pdf_in.**PDFIndexingConverter**
Bases: object

class MoinMoin.converter.pdf_in.**UnicodeConverter**(*rsrcmgr*, *pageno=1*, *laparams=None*, *showpageno=False*)
Bases: pdfminer.converter.TextConverter

read_result()

write_text(*text*)

MoinMoin.converter.pygments_in module

MoinMoin - Pygments driven syntax highlighting input converter

class MoinMoin.converter.pygments_in.**Converter**(*lexer=None*, *contenttype=None*)
Bases: object

class MoinMoin.converter.pygments_in.**TreeFormatter**(***options*)
Bases: pygments.formatter.Formatter

format(*tokensource*, *element*)

MoinMoin.converter.rst_in module

MoinMoin - ReStructured Text input converter

It's based on docutils rst parser. Conversion of docutils document tree to moinmoin document tree.

This converter based on ReStructuredText 2006-09-22. Works with docutils version 0.5 (2008-06-25) or higher.

class MoinMoin.converter.rst_in.**Converter**
Bases: object

classmethod factory(*input*, *output*, ***kw*)

class MoinMoin.converter.rst_in.**MoinDirectives**
Bases: object

Class to handle all custom directive handling. This code is called as part of the parsing stage.

include(*name*, *arguments*, *options*, *content*, *lineno*, *content_offset*, *block_text*, *state*, *state_machine*)

macro(*name*, *arguments*, *options*, *content*, *lineno*, *content_offset*, *block_text*, *state*, *state_machine*)

parser(*name*, *arguments*, *options*, *content*, *lineo*, *content_offset*, *block_text*, *state*, *state_machine*)

table_of_content(*name*, *arguments*, *options*, *content*, *lineno*, *content_offset*, *block_text*, *state*, *state_machine*)

class MoinMoin.converter.rst_in.**NodeVisitor**
Bases: object

Part of docutils which converts docutils DOM tree to Moin DOM tree

close_moin_page_node()

depart_Text(*node*)

depart_admonition(*node=None*)

depart_attention(*node=None*)

depart_attribution(*node*)

depart_author(*node*)

depart_block_quote(*node*)

depart_bullet_list(*node*)

depart_caution(*node=None*)

depart_comment(*node*)

depart_copyright(*node*)

depart_danger(*node=None*)

depart_definition(*node*)

depart_definition_list(*node*)

depart_definition_list_item(*node*)

depart_description(*node*)

depart_docinfo(*node*)

depart_emphasis(*node*)

depart_entry(*node*)

depart_enumerated_list(*node*)

depart_error(*node=None*)

depart_field(*node*)

depart_field_body(*node*)

depart_field_list(*node*)

depart_field_name(*node*)

depart_figure(*node*)

depart_footer(*node*)

depart_footnote(*node*)

depart_footnote_reference(*node*)

depart_header(*node*)

depart_hint(*node=None*)

depart_image(*node*)

depart_important(*node=None*)

depart_inline(*node*)

depart_label(*node*)

depart_line(*node*)

depart_line_block(*node*)

depart_list_item(*node*)

depart_literal_block(*node*)

depart_note(*node=None*)

depart_option(*node*)

depart_option_list(*node*)

depart_option_list_item(*node*)

depart_paragraph(*node*)

depart_problematic(*node*)

depart_reference(*node*)

depart_row(*node*)

depart_rubric(*node*)

depart_section(*node*)

depart_sidebar(*node*)

depart_strong(*node*)

depart_subscript(*node*)

depart_substitution_definition(*node*)

depart_subtitle(*node*)

depart_superscript(*node*)

depart_system_message(*node*)

depart_table(*node*)

depart_target(*node*)

depart_tbody(*node*)

depart_term(*node*)

depart_tgroup(*node*)

depart_thead(*node*)

depart_tip(*node=None*)

depart_title(*node*)

depart_title_reference(*node*)

depart_topic(*node*)

depart_transition(*node*)

depart_version(*node*)

depart_warning(*node=None*)

dispatch_departure(*node*)
Call self."depart_ + node class name" with *node* as parameter. If the depart_... method does not exist, call self.unknown_departure.

dispatch_visit(*node*)
Call self."visit_ + node class name" with *node* as parameter. If the visit_... method does not exist, call self.unknown_visit.

open_moin_page_node(*mointree_element*)

tree()

unimplemented_visit(*node*)

unknown_departure(*node*)
Called before exiting unknown *Node* types.

Raise exception unless overridden.

unknown_visit(*node*)
Called when entering unknown *Node* types.

Raise an exception unless overridden.

visit_Text(*node*)

visit_admonition(*node*)

visit_attention(*node*)

visit_attribution(*node*)

visit_author(*node*)

visit_block_quote(*node*)

visit_bullet_list(*node*)

visit_caution(*node*)

visit_comment(*node*)
Create moinwiki style hidden comment rather than html style: <!– a comment –>

visit_copyright(*node*)

visit_danger(*node*)

visit_definition(*node*)

visit_definition_list(*node*)

visit_definition_list_item(*node*)

visit_description(*node*)

visit_docinfo(*node*)

visit_emphasis(*node*)

visit_entry(*node*)

visit_enumerated_list(*node*)

visit_error(*node*)

visit_field(*node*)

visit_field_body(*node*)

visit_field_list(*node*)

visit_field_name(*node*)

visit_figure(*node*)

visit_footer(*node*)

visit_footnote(*node*)

visit_footnote_reference(*node*)

visit_header(*node*)

visit_hint(*node*)

visit_image(*node*)
Processes images and other transcluded objects.

visit_important(*node*)

visit_inline(*node*)

visit_label(*node*)

visit_line(*node*)

visit_line_block(*node*)

visit_list_item(*node*)

visit_literal(*node*)

visit_literal_block(*node*)

visit_note(*node*)

visit_option(*node*)

visit_option_list(*node*)

visit_option_list_item(*node*)

visit_paragraph(*node*)

visit_problematic(*node*)

visit_reference(*node*)

visit_row(*node*)

visit_rubric(*node*)

visit_section(*node*)

visit_sidebar(*node*)

visit_strong(*node*)

visit_subscript(*node*)

visit_substitution_definition(*node*)

visit_subtitle(*node*)

visit_superscript(*node*)

visit_system_message(*node*)

visit_table(*node*)

visit_target(*node*)

visit_tbody(*node*)

visit_term(*node*)

visit_tgroup(*node*)
The tgroup node is presented as the parent of thead and tbody. These should be siblings. Other children are colspec which have a colwidth attribute. Using these numbers to specify a width on the col element similar to Sphinx results in an HTML validation error. There is no markup to specify styling such as background color. Best result is to discard this node.

visit_thead(*node*)

visit_tip(*node*)

visit_title(*node*)

visit_title_reference(*node*)

visit_topic(*node*)

visit_transition(*node*, *default_class=u'moin-hr3'*)

visit_version(*node*)

visit_warning(*node*)

class MoinMoin.converter.rst_in.**Writer**
Bases: docutils.writers.Writer

config_section = 'MoinMoin writer'

config_section_dependencies = ('writers',)

output = None

supported = ('moin-x-document',)

translate()

visitor_attributes = []

MoinMoin.converter.rst_in.**walkabout**(*node*, *visitor*)
This is tree traversal part of docutils without docutils logging.

MoinMoin.converter.rst_out module

MoinMoin - reStructuredText markup output converter

Converts an internal document tree into reStructuredText markup.

This converter based on ReStructuredText 2006-09-22.

class MoinMoin.converter.rst_out.**Cell**(*text*)
Bases: object

height()

width()

class MoinMoin.converter.rst_out.**Converter**
Bases: object

Converter application/x.moin.document -> text/x.moin.rst

define_references()
Adds defenitions of founded links and objects to the converter output.

classmethod factory(*input*, *output*, ***kw*)

namespaces = {<Namespace('http://moinmo.in/namespaces/page')>: 'moinpage'}

open(*elem*)

open_children(*elem*)

open_moinpage(*elem*)

open_moinpage_a(*elem*)

open_moinpage_blockcode(*elem*)

open_moinpage_body(*elem*)

open_moinpage_code(*elem*)

open_moinpage_emphasis(*elem*)

open_moinpage_h(*elem*)

open_moinpage_inline_part(*elem*)

open_moinpage_line_break(*elem*)

open_moinpage_list(*elem*)

open_moinpage_list_item(*elem*)

open_moinpage_list_item_body(*elem*)

open_moinpage_list_item_label(*elem*)

open_moinpage_note(*elem*)

open_moinpage_object(*elem*)

open_moinpage_p(*elem*)

open_moinpage_page(*elem*)

open_moinpage_part(*elem*)

open_moinpage_separator(*elem*)

open_moinpage_span(*elem*)

open_moinpage_strong(*elem*)

open_moinpage_table(*elem*)

open_moinpage_table_body(*elem*)

open_moinpage_table_cell(*elem*)

open_moinpage_table_header(*elem*)

open_moinpage_table_of_content(*elem*)

open_moinpage_table_row(*elem*)

supported_tag = {'moinpage': ('a', 'blockcode', 'break_line', 'code', 'div', 'emphasis', 'h', 'list', 'list_item', 'list_i

class MoinMoin.converter.rst_out.**ReST**
Bases: object

ReST syntax elements It's dummy

a_separator = u'|'

emphasis = u'*'

h = [u'=', u'-', u'`', u':', u'"', u'"', u'~', u'^', u'_', u'*', u'+', u'#', u'<', u'>']

linebreak = u'\n\n'

list_type = {(u'ordered', None): u'1.', (u'ordered', u'upper-alpha'): u'A.', (u'unordered', None): u'*', (u'definitio

monospace = u'``'

p = u'\n'

separator = u'----'

strong = u''**

verbatim = u'::'

class MoinMoin.converter.rst_out.**Table**
Bases: object

An object of this class collects the structure of a table and represent it in ReStructuredText syntax.

add_cell(*cs*, *rs*, *cell*)
Adds cell to the row.

Parameters **cs** – number of columns spanned

add_row()
Add new row to the table.

col_width(*col*)
Counts the width of the column in ReSturcturedText representation.

Parameters **col** – index of the column

Returns number of characters

end_row()
Adds empyt cells to current row if it's too short.

Moves the row to the head of the table if it is table header.

height()

Returns number of rows in the table

row_height(*row*)
Counts lines in ReSturcturedText representation of the row

Parameters **row** – index of the row

Returns number of lines

width()

Returns width of rows in the table or zero if rows have different width

MoinMoin.converter.smiley module

MoinMoin - Smiley converter

Replace all the text corresponding to a smiley, by the corresponding element for the DOM Tree.

class MoinMoin.converter.smiley.**Converter**
Bases: object

Replace each smiley by the corresponding element in the DOM Tree

do_children(*element*)

do_smiley(*element*)
From a text, return a list with smileys replaced by the appropriate elements, and the former text for the other elements of the list.

replace_smiley(*text*)
Replace a given string by the appropriate element if the string is exactly a smiley. Otherwise return the string without any change.

s = ':))'

smiley_re = <_sre.SRE_Pattern object at 0x466d310>

smiley_rule = u'\n (^|(?<=\\s)) # we require either beginning of line or some space before a smiley\n (\\<\\:\\(|X\\-\\(

smileys = {'<:(': 'frown', 'X-(': 'angry', ':)': 'smile', ':-))': 'smile3', ':(': 'sad', '/!\\': 'alert', '{X}': 'icon-error', '{C

tags_to_ignore = set(['blockcode', 'code', 'nowiki'])

MoinMoin.converter.text_csv_in module

MoinMoin - CSV text data to DOM converter

class MoinMoin.converter.text_csv_in.**Converter**
Bases: MoinMoin.converter._table.TableMixin

Parse the raw text and create a document object that can be converted into output using Emitter.

MoinMoin.converter.text_in module

MoinMoin - Simple text input converter.

It just puts all text into a code block. It acts as a wildcard for text/* input.

We keep it at MIDDLE+2 prio in the registry, one after pygments converter, so it is a fallback for the case we have no pygments or pygments has no support for the input mimetype.

class MoinMoin.converter.text_in.**Converter**
Bases: object

Parse the raw text and create a document object that can be converted into output using Emitter.

MoinMoin.converter.text_out module

MoinMoin - plain text output converter

Converts an internal document tree into plain, unformatted text.

The purpose of this converter is mainly to be used in a converter chain like markup -> dom -> text and get rid of the (wiki, rst, docbook, ...) markup that way, so we get indexable plain text for our search index.

class MoinMoin.converter.text_out.**Converter**
Bases: object

Converter application/x.moin.document -> text/plain

classmethod factory(*input*, *output*, ***kw*)

MoinMoin.converter.xml_in module

MoinMoin - Generic XML input converter

class MoinMoin.converter.xml_in.**XMLIndexingConverter**
Bases: object

We try to generically extract contents from XML documents by just throwing away all XML tags. This is for indexing, so this might be good enough.

MoinMoin.converter.xml_in.**strip_xml**(*text*)

Module contents

MoinMoin - Converter support

Converters are used to convert between formats or between different featuresets of one format.

There are usually three types of converters:

- Between an input format like Moin Wiki or Creole and the internal tree representation.
- Between the internal tree and an output format like HTML.
- Between different featuresets of the internal tree representation like URI types or macro expansion.

TODO: Merge with new-style macros.

class MoinMoin.converter.**RegistryConverter**
Bases: *MoinMoin.util.registry.RegistryBase*

class Entry
Bases: MoinMoin.converter.Entry

RegistryConverter.**register**(*factory*, *type_input*, *type_output*, *priority=0*)
Register a factory

Parameters **factory** – Factory to register. Callable, must return an object.

MoinMoin.datastruct package

Subpackages

MoinMoin.datastruct.backends package

Submodules

MoinMoin.datastruct.backends.composite_dicts module

MoinMoin - dict access via various backends.

class MoinMoin.datastruct.backends.composite_dicts.**CompositeDicts**(**backends*)
Bases: *MoinMoin.datastruct.backends.BaseDictsBackend*

Manage several dicts backends.

MoinMoin.datastruct.backends.composite_groups module

MoinMoin - group access via various backends.

The composite_groups is a backend that does not have direct storage, but composes other backends to a new one, so group definitions are retrieved from several backends. This allows to mix different backends.

class MoinMoin.datastruct.backends.composite_groups.**CompositeGroups**(**backends*)
Bases: *MoinMoin.datastruct.backends.BaseGroupsBackend*

Manage several group backends.

MoinMoin.datastruct.backends.config_dicts module

MoinMoin - config dict backend

The config group backend enables you to define dicts in a configuration file.

class MoinMoin.datastruct.backends.config_dicts.**ConfigDict**(*name*, *backend*)
Bases: *MoinMoin.datastruct.backends.BaseDict*

class MoinMoin.datastruct.backends.config_dicts.**ConfigDicts**(*dicts*)
Bases: *MoinMoin.datastruct.backends.BaseDictsBackend*

MoinMoin.datastruct.backends.config_groups module

MoinMoin - config groups backend

The config_groups backend enables one to define groups and their members in a configuration file.

class MoinMoin.datastruct.backends.config_groups.**ConfigGroup**(*name*, *backend*)
Bases: *MoinMoin.datastruct.backends.GreedyGroup*

class MoinMoin.datastruct.backends.config_groups.**ConfigGroups**(*groups*)
Bases: *MoinMoin.datastruct.backends.BaseGroupsBackend*

MoinMoin.datastruct.backends.config_lazy_groups module

MoinMoin - config group lazy backend.

The config group backend allows one to define groups in a configuration file.

NOTE that this is proof-of-concept implementation. LDAP backend should be based on this concept.

class MoinMoin.datastruct.backends.config_lazy_groups.**ConfigLazyGroup**(*name*, *backend*)
Bases: *MoinMoin.datastruct.backends.LazyGroup*

class MoinMoin.datastruct.backends.config_lazy_groups.**ConfigLazyGroups**(*groups*)
Bases: *MoinMoin.datastruct.backends.LazyGroupsBackend*

MoinMoin.datastruct.backends.wiki_dicts module

MoinMoin - WikiDict functions.

class MoinMoin.datastruct.backends.wiki_dicts.**WikiDict**(*name*, *backend*)
Bases: *MoinMoin.datastruct.backends.BaseDict*

Mapping of keys to values from meta of an item.

class MoinMoin.datastruct.backends.wiki_dicts.**WikiDicts**
Bases: *MoinMoin.datastruct.backends.BaseDictsBackend*

MoinMoin.datastruct.backends.wiki_groups module

MoinMoin - wiki group backend

The wiki_groups backend allows to define groups on wiki items.

Normally, the name of the group item has to end with Group like FriendsGroup. This lets MoinMoin recognize it as a group. This default pattern could be changed (e.g. for non-english languages etc.), see HelpOnConfiguration.

class MoinMoin.datastruct.backends.wiki_groups.**WikiGroup**(*name*, *backend*)
Bases: *MoinMoin.datastruct.backends.GreedyGroup*

class MoinMoin.datastruct.backends.wiki_groups.**WikiGroups**
Bases: *MoinMoin.datastruct.backends.BaseGroupsBackend*

Module contents

MoinMoin - base classes for datastructs.

TODO: Per https://docs.python.org/2/library/userdict.html Starting with Python version 2.6, it is recommended to use collections.MutableMapping instead of DictMixin.

class MoinMoin.datastruct.backends.**BaseDict**(*name*, *backend*)
Bases: object, UserDict.DictMixin

get(*key*, *default=None*)
Return the value if key is in the dictionary, else default. If default is not given, it defaults to None, so that this method never raises a KeyError.

keys()

class MoinMoin.datastruct.backends.**BaseDictsBackend**
Bases: object

get(*key*, *default=None*)
Return the dictionary named <key> if key is in the backend, else default. If default is not given, it defaults to None, so that this method never raises a DictDoesNotExistError.

is_dict_name(*name*)

class MoinMoin.datastruct.backends.**BaseGroup**(*name*, *backend*)
Bases: object

Group is something which stores members. Groups are immutable. A member is some arbitrary entity name (Unicode object).

class MoinMoin.datastruct.backends.**BaseGroupsBackend**
Bases: object

Backend provides access to the group definitions for the other MoinMoin code.

get(*key*, *default=None*)
Return the group named <key> if key is in the backend, else default. If default is not given, it defaults to None, so that this method never raises a GroupDoesNotExistError.

groups_with_member(*member*)
List all group names of groups containing <member>.

Parameters **member** – member name [unicode]

Returns list of group names [unicode]

is_group_name(*member*)

exception MoinMoin.datastruct.backends.**DictDoesNotExistError**
Bases: exceptions.Exception

Raised when a dict name is not found in the backend.

class MoinMoin.datastruct.backends.**GreedyGroup**(*name*, *backend*)
Bases: *MoinMoin.datastruct.backends.BaseGroup*

GreedyGroup gets all members during initialization and stores them internally.

Members of a group may be names of other groups.

exception MoinMoin.datastruct.backends.**GroupDoesNotExistError**
Bases: exceptions.Exception

Raised when a group name is not found in the backend.

class MoinMoin.datastruct.backends.**LazyGroup**(*name*, *backend*)
Bases: *MoinMoin.datastruct.backends.BaseGroup*

A lazy group does not store members internally, but gets them from a backend when needed.

Lazy group is made only of members. It can not consist of other groups.

For instance, this is a possible LazyGroup:

PossibleGroup

- OneMember
- OtherMember

This is a group which cannot be LazyGroup:

NotPossibleGroup

- OneMember
- OtherMember
- OtherGroup

class MoinMoin.datastruct.backends.**LazyGroupsBackend**
Bases: *MoinMoin.datastruct.backends.BaseGroupsBackend*

Module contents

MoinMoin - datastruct (groups and dicts) support.

MoinMoin.i18n package

Module contents

MoinMoin - i18n (internationalization) and l10n (localization) support

To use this, please use exactly this line (no less, no more):

```
from MoinMoin.i18n import _, L_, N_

# _ == gettext
# N_ == ngettext
# L_ == lazy_gettext
```

MoinMoin.i18n.**force_locale**(**args*, ***kwds*)
Temporarily overrides the currently selected locale. Sometimes it is useful to switch the current locale to different one, do some tasks and then revert back to the original one. For example, if the user uses German on the web site, but you want to send them an email in English, you can use this function as a context manager:

```
with force_locale('en_US'):
    send_email(gettext('Hello!'), ...)
```

MoinMoin.i18n.**get_locale**()
return the locale for the current user

MoinMoin.i18n.**get_timezone**()
return the timezone for the current user

MoinMoin.i18n.**i18n_init**(*app*)
initialize Flask-Babel

MoinMoin.items package

Subpackages

Submodules

MoinMoin.items.blog module

MoinMoin - Blog itemtype

class MoinMoin.items.blog.**Blog**(*fqname*, *rev=None*, *content=None*)
Bases: *MoinMoin.items.Default*

description = l'Blog item'

display_name = l'Blog'

do_show(*revid*)
Show a blog item and a list of its blog entries below it.

If tag GET-parameter is defined, the list of blog entries consists only of those entries that contain the tag value in their lists of tags.

itemtype = u'blog'

order = 0

class MoinMoin.items.blog.**BlogEntry**(*fqname*, *rev=None*, *content=None*)
Bases: *MoinMoin.items.Default*

description = l'Blog entry item'

display_name = l'Blog entry'

do_show(*revid*)

itemtype = u'blogentry'

order = 0

class MoinMoin.items.blog.**BlogEntryMetaForm**(*value=Unspecified*, ***kw*)
Bases: *MoinMoin.items.BaseMetaForm*

field_schema = [<class 'flatland.schema.declarative.String'>, <class 'flatland.schema.declarative.String'>, <class '

class MoinMoin.items.blog.**BlogMetaForm**(*value=Unspecified*, ***kw*)
Bases: *MoinMoin.items.BaseMetaForm*

field_schema = [<class 'flatland.schema.declarative.String'>, <class 'flatland.schema.declarative.String'>, <class '

MoinMoin.items.content module

MoinMoin - item contents

Classes handling the content part of items (ie. minus metadata). The content part is sometimes called the "data" part in other places, but is always called content in this module to avoid confusion.

Each class in this module corresponds to a contenttype value.

class MoinMoin.items.content.**AnyWikiDraw**(*contenttype*, *item=None*)
Bases: *MoinMoin.items.content.DrawAWDTWDBase*

drawings by AnyWikiDraw applet. It creates three files which are stored as tar file.

class ModifyForm(*value=Unspecified*, ***kw*)
Bases: MoinMoin.items.content.ModifyForm

field_schema = [<class 'flatland.schema.declarative.FileStorage'>]

template = 'modify_anywikidraw.html'

AnyWikiDraw.**contenttype = 'application/x-anywikidraw'**

AnyWikiDraw.**display_name = 'ADRAW'**

class MoinMoin.items.content.**Application**(*contenttype*, *item=None*)
Bases: *MoinMoin.items.content.Binary*

Base class for application/*

class MoinMoin.items.content.**ApplicationXGTar**(*contenttype*, *item=None*)
Bases: *MoinMoin.items.content.ApplicationXTar*

Compressed tar items

contenttype = 'application/x-gtar'

display_name = 'TGZ'

class MoinMoin.items.content.**ApplicationXTar**(*contenttype*, *item=None*)
Bases: *MoinMoin.items.content.TarMixin*, *MoinMoin.items.content.Application*

Tar items

contenttype = 'application/x-tar'

display_name = 'TAR'

class MoinMoin.items.content.**ApplicationZip**(*contenttype*, *item=None*)
Bases: *MoinMoin.items.content.ZipMixin*, *MoinMoin.items.content.Application*

Zip items

contenttype = 'application/zip'

display_name = 'ZIP'

class MoinMoin.items.content.**Audio**(*contenttype*, *item=None*)
Bases: *MoinMoin.items.content.Binary*

Base class for audio/*

contenttype = 'audio/*'

group = 'Audio Items'

class MoinMoin.items.content.**Binary**(*contenttype*, *item=None*)
Bases: *MoinMoin.items.content.Content*

An arbitrary binary item, fallback class for every item mimetype.

class **ModifyForm**(*value=Unspecified*, ***kw*)
Bases: flatland.schema.declarative.Form

The content part of the ModifyForm of an Item subclass. See also the doc of Item._ModifyForm.

field_schema = [<class 'flatland.schema.declarative.FileStorage'>]

template = 'modify_binary.html'

Binary.**contenttype = '*/*'**

Binary.**data**

Binary.**do_get**(*force_attachment=False*, *mimetype=None*)

Binary.**get_data**()

class MoinMoin.items.content.**CSV**(*contenttype*, *item=None*)
Bases: *MoinMoin.items.content.Text*

contenttype = 'text/csv'

display_name = 'CSV'

class MoinMoin.items.content.**Content**(*contenttype*, *item=None*)
Bases: object

Base for content classes defining some helpers, agnostic about content data.

contenttype = None

classmethod create(*contenttype*, *item=None*)

data

default_contenttype_params = {}

display_name = None

get_data()

get_templates(*contenttype=None*)
create a list of templates (for some specific contenttype)

group = 'Other Items'

ingroup_order = 0

internal_representation(**args*, ***kw*)
Return the internal representation of a document using a DOM Tree

name

rev

class MoinMoin.items.content.**CreoleWiki**(*contenttype*, *item=None*)
Bases: *MoinMoin.items.content.MarkupItem*

Creole wiki markup

contenttype = 'text/x.moin.creole'

display_name = 'Wiki (Creole)'

class MoinMoin.items.content.**Diff**(*contenttype*, *item=None*)
Bases: *MoinMoin.items.content.Text*

contenttype = 'text/x-diff'

display_name = 'Diff/Patch'

class MoinMoin.items.content.**DocBook**(*contenttype*, *item=None*)
Bases: *MoinMoin.items.content.MarkupItem*

DocBook Document

contenttype = 'application/docbook+xml'

display_name = 'DocBook'

class MoinMoin.items.content.**Draw**(*contenttype*, *item=None*)
Bases: *MoinMoin.items.content.TarMixin*, *MoinMoin.items.content.Image*

Base class for drawing apps that use special Java/Javascript applets to modify and store data in a tar file.

class ModifyForm(*value=Unspecified*, ***kw*)
Bases: MoinMoin.items.content.ModifyForm

field_schema = [<class 'flatland.schema.declarative.FileStorage'>]

is_draw = True

Draw.**group = 'Drawing Items'**

Draw.**handle_post**()

class MoinMoin.items.content.**DrawAWDTWDBase**(*contenttype*, *item=None*)
Bases: *MoinMoin.items.content.DrawPNGMap*

Shared code between TWikiDraw and AnyWikiDraw

handle_post()

class MoinMoin.items.content.**DrawPNGMap**(*contenttype*, *item=None*)
Bases: *MoinMoin.items.content.Draw*

Base class for drawings that have a png with click map

class MoinMoin.items.content.**GIF**(*contenttype*, *item=None*)
Bases: *MoinMoin.items.content.TransformableBitmapImage*

GIF image.

contenttype = 'image/gif'

display_name = 'GIF'

class MoinMoin.items.content.**HTML**(*contenttype*, *item=None*)
Bases: *MoinMoin.items.content.MarkupItem*

HTML markup

Note: As we use html_in converter to convert this to DOM and later some output converterter to produce output format (e.g. html_out for html output), all(?) unsafe stuff will get lost.

Note: If raw revision data is accessed, unsafe stuff might be present!

class ModifyForm(*value=Unspecified*, ***kw*)
Bases: MoinMoin.items.content.ModifyForm

field_schema = [<class 'flatland.schema.declarative.FileStorage'>, <class 'flatland.schema.declarative.String'

template = 'modify_text_html.html'

HTML.**contenttype = 'text/html'**

HTML.**display_name = 'HTML'**

class MoinMoin.items.content.**IRCLog**(*contenttype*, *item=None*)
Bases: *MoinMoin.items.content.Text*

contenttype = 'text/x-irclog'

display_name = 'IRC Log'

class MoinMoin.items.content.**Image**(*contenttype*, *item=None*)
Bases: *MoinMoin.items.content.Binary*

Base class for image/*

contenttype = 'image/*'

class MoinMoin.items.content.**JPEG**(*contenttype*, *item=None*)
Bases: *MoinMoin.items.content.TransformableBitmapImage*

JPEG image.

contenttype = 'image/jpeg'

display_name = 'JPEG'

class MoinMoin.items.content.**MP3**(*contenttype*, *item=None*)
Bases: *MoinMoin.items.content.Audio*

contenttype = 'audio/mpeg'

display_name = 'MP3'

class MoinMoin.items.content.**MP4**(*contenttype*, *item=None*)
Bases: *MoinMoin.items.content.Video*

contenttype = 'video/mp4'

display_name = 'MP4'

class MoinMoin.items.content.**Markdown**(*contenttype*, *item=None*)
Bases: *MoinMoin.items.content.MarkupItem*

Markdown markup

contenttype = 'text/x-markdown'

display_name = 'Markdown'

class MoinMoin.items.content.**MarkupItem**(*contenttype*, *item=None*)
Bases: *MoinMoin.items.content.Text*

some kind of item with markup (internal links and transcluded items)

group = 'Markup Text Items'

class MoinMoin.items.content.**MediaWiki**(*contenttype*, *item=None*)
Bases: *MoinMoin.items.content.MarkupItem*

MediaWiki markup

contenttype = 'text/x-mediawiki'

display_name = 'Wiki (MediaWiki)'

class MoinMoin.items.content.**MoinWiki**(*contenttype*, *item=None*)
Bases: *MoinMoin.items.content.MarkupItem*

MoinMoin wiki markup

contenttype = 'text/x.moin.wiki'

display_name = 'Wiki (MoinMoin)'

class MoinMoin.items.content.**NonExistentContent**(*contenttype*, *item=None*)
Bases: *MoinMoin.items.content.Content*

Dummy Content to use with NonExistent.

contenttype = u'application/x-nonexistent'

do_get(*force_attachment=False*, *mimetype=None*)

group = None

class MoinMoin.items.content.**OGGAudio**(*contenttype*, *item=None*)
Bases: *MoinMoin.items.content.Audio*

contenttype = 'audio/ogg'

display_name = 'OGG'

class MoinMoin.items.content.**OGGVideo**(*contenttype*, *item=None*)
Bases: *MoinMoin.items.content.Video*

contenttype = 'video/ogg'

display_name = 'OGG'

class MoinMoin.items.content.**OctetStream**(*contenttype*, *item=None*)
Bases: *MoinMoin.items.content.Binary*

Fallback Content for uploaded file of unknown contenttype.

contenttype = 'application/octet-stream'

display_name = 'Binary File'

class MoinMoin.items.content.**PDF**(*contenttype*, *item=None*)
Bases: *MoinMoin.items.content.Application*

contenttype = 'application/pdf'

display_name = 'PDF'

class MoinMoin.items.content.**PNG**(*contenttype*, *item=None*)
Bases: *MoinMoin.items.content.TransformableBitmapImage*

PNG image.

contenttype = 'image/png'

display_name = 'PNG'

class MoinMoin.items.content.**PlainText**(*contenttype*, *item=None*)
Bases: *MoinMoin.items.content.Text*

contenttype = 'text/plain'

display_name = 'Plain Text'

class MoinMoin.items.content.**PythonCode**(*contenttype*, *item=None*)
Bases: *MoinMoin.items.content.Text*

contenttype = 'text/x-python'

display_name = 'Python Code'

class MoinMoin.items.content.**ReST**(*contenttype*, *item=None*)
Bases: *MoinMoin.items.content.MarkupItem*

ReStructured Text markup

contenttype = 'text/x-rst'

display_name = 'ReST'

class MoinMoin.items.content.**RegistryContent**(*group_names*)
Bases: *MoinMoin.util.registry.RegistryBase*

class Entry
Bases: MoinMoin.items.content.Entry

RegistryContent.**register**(*e*, *group*)
Register a contenttype entry and optionally add it to a specific group.

class MoinMoin.items.content.**RenderableBinary**(*contenttype*, *item=None*)
Bases: *MoinMoin.items.content.Binary*

Base class for some binary stuff that renders with a object tag.

class MoinMoin.items.content.**RenderableBitmapImage**(*contenttype*, *item=None*)
Bases: *MoinMoin.items.content.RenderableImage*

PNG/JPEG/GIF images use <img> tag (better browser support than <object>)

class MoinMoin.items.content.**RenderableImage**(*contenttype*, *item=None*)
Bases: *MoinMoin.items.content.RenderableBinary*

Base class for renderable Image mimetypes

group = 'Image Items'

class MoinMoin.items.content.**SvgDraw**(*contenttype*, *item=None*)
Bases: *MoinMoin.items.content.Draw*

drawings by svg-edit. It creates two files (svg, png) which are stored as tar file.

class ModifyForm(*value=Unspecified*, ***kw*)
Bases: MoinMoin.items.content.ModifyForm

field_schema = [<class 'flatland.schema.declarative.FileStorage'>]

template = 'modify_svg-edit.html'

SvgDraw.**contenttype = 'application/x-svgdraw'**

SvgDraw.**display_name = 'SVGDRAW'**

SvgDraw.**handle_post**()

class MoinMoin.items.content.**SvgImage**(*contenttype*, *item=None*)
Bases: *MoinMoin.items.content.RenderableImage*

SVG images use <object> tag mechanism from RenderableBinary base class

contenttype = 'image/svg+xml'

display_name = 'SVG'

class MoinMoin.items.content.**TWikiDraw**(*contenttype*, *item=None*)
Bases: *MoinMoin.items.content.DrawAWDTWDBase*

drawings by TWikiDraw applet. It creates three files which are stored as tar file.

class ModifyForm(*value=Unspecified*, ***kw*)
Bases: MoinMoin.items.content.ModifyForm

field_schema = [<class 'flatland.schema.declarative.FileStorage'>]

template = 'modify_twikidraw.html'

TWikiDraw.**contenttype = 'application/x-twikidraw'**

TWikiDraw.**display_name = 'TDRAW'**

class MoinMoin.items.content.**TarMixin**
Bases: object

TarMixin offers additional functionality for tar-like items to list and access member files and to create new revisions by multiple posts.

get_member(*name*)
return a file-like object with the member file data

Parameters **name** – name of the data in the container file

list_members()
list tar file contents (member file names)

put_member(*name*, *content*, *content_length*, *expected_members*)
puts a new member file into a temporary tar container. If all expected members have been put, it saves the tar container to a new item revision.

Parameters

- **name** – name of the data in the container file
- **content** – the data to store into the tar file (str or file-like)
- **content_length** – byte-length of content (for str, None can be given)
- **expected_members** – set of expected member file names

class MoinMoin.items.content.**Text**(*contenttype*, *item=None*)
Bases: *MoinMoin.items.content.Binary*

Base class for text/*

class **ModifyForm**(*value=Unspecified*, ***kw*)
Bases: MoinMoin.items.content.ModifyForm

cols = 80

field_schema = [<class 'flatland.schema.declarative.FileStorage'>, <class 'flatland.schema.declarative.String'

rows = 20

template = 'modify_text.html'

Text.**contenttype = 'text/*'**

Text.**data_form_to_internal**(*data*)
convert data from form format to memory format

Text.**data_internal_to_form**(*text*)
convert data from memory format to form format

Text.**data_internal_to_storage**(*text*)
convert data from memory format to storage format

Text.**data_storage_to_internal**(*data*)
convert data from storage format to memory format

Text.**default_contenttype_params = {'charset': 'utf-8'}**

Text.**group = 'Other Text Items'**

class MoinMoin.items.content.**TransformableBitmapImage**(*contenttype*, *item=None*)
Bases: *MoinMoin.items.content.RenderableBitmapImage*

We can transform (resize, rotate, mirror) some image types

class MoinMoin.items.content.**Video**(*contenttype*, *item=None*)
Bases: *MoinMoin.items.content.Binary*

Base class for video/*

contenttype = 'video/*'

group = 'Video Items'

class MoinMoin.items.content.**WAV**(*contenttype*, *item=None*)
Bases: *MoinMoin.items.content.Audio*

contenttype = 'audio/x-wav'

display_name = 'WAV'

class MoinMoin.items.content.**WebMAudio**(*contenttype*, *item=None*)
Bases: *MoinMoin.items.content.Audio*

contenttype = 'audio/webm'

display_name = 'WebM'

class MoinMoin.items.content.**WebMVideo**(*contenttype*, *item=None*)
Bases: *MoinMoin.items.content.Video*

contenttype = 'video/webm'

display_name = 'WebM'

class MoinMoin.items.content.**ZipMixin**
Bases: object

ZipMixin offers additional functionality for zip-like items to list and access member files.

get_member(*name*)
return a file-like object with the member file data

Parameters **name** – name of the data in the zip file

list_members()
list zip file contents (member file names)

put_member(*name*, *content*, *content_length*, *expected_members*)

MoinMoin.items.content.**conv_serialize**(*doc*, *namespaces*, *method='polyglot'*)

MoinMoin.items.content.**register**(*cls*)

MoinMoin.items.ticket module

MoinMoin - Ticket itemtype

TODO: Tickets require more work. Key requirements include but are not limited to:

- ability to edit the original description or subsequent comments to correct typos
- rework global history and global index to show or not show tickets and/or comments
- rework page trail, subscriptions, navigation links to show something other than rev ids or timestamps
- some method to override the default parser (text/x.moin.wiki) for a new ticket or comment
- add more comments to the code

A ticket is a unique itemtype, the initial ticket consists of a description and meta data. Some of the meta data fields are unique to tickets.

The original design called for tickets to be nameless. Instead of a name, tickets would have a summary in meta data that would be too long for a typical wiki item name. However, it is not currently possible to create a nameless item, so as a workaround, tickets are created with a name similar to: ticket_2016_09_21-09_19_29.

When a ticket's meta data is updated, a new revision is created and the ticket's name is removed. After the first update, the current revision has an Old Name, after the second meta data update, the current name and Old Name are both displayed as None.

Ticket comments and ticket comments to comments are presently created without an item type. This is an issue because when a wiki is dumped and restored, all comments are tagged with an itemtype of "default". This is probably not wanted, but no other problems after a restore were noted.

As a workaround to the issue of creating nameless items, comments are created with a name similar to: comment_2016_09_21-09_19_29. As comments can not currently be updated, the name is never removed. Comments and comments to comments are linked to the original ticket through a refers_to field preserved in meta data.

class MoinMoin.items.ticket.**AdvancedSearchForm**(*value=Unspecified*, ***kw*)
Bases: flatland.schema.declarative.Form

field_schema = [<class 'flatland.schema.declarative.Integer'>, <class 'flatland.schema.declarative.Integer'>, <class

MoinMoin.items.ticket.**OptionalTicketReference**
alias of Reference

MoinMoin.items.ticket.**OptionalUserReference**
alias of Reference

MoinMoin.items.ticket.**Rating**
alias of Integer

class MoinMoin.items.ticket.**Ticket**(*fqname*, *rev=None*, *content=None*)
Bases: *MoinMoin.items.Contentful*

description = l'Ticket item'

display_name = l'Ticket'

do_modify()
Process new ticket, changes to ticket meta data, and/or a new comment against original ticket description.

User has clicked "Submit ticket" or "Update ticket" button to get here. If user clicks Save button to add a comment to a prior comment it is not processed here - see /+comment in views.py.

do_show(*revid*)

itemtype = u'ticket'

modify_template = 'ticket/modify.html'

submit_template = 'ticket/submit.html'

class MoinMoin.items.ticket.**TicketBackRefForm**(*value=Unspecified*, ***kw*)
Bases: flatland.schema.declarative.Form

field_schema = [<class 'flatland.schema.declarative.BackReference'>, <class 'flatland.schema.declarative.BackRef

class MoinMoin.items.ticket.**TicketForm**(*value=Unspecified*, ***kw*)
Bases: *MoinMoin.items.BaseModifyForm*

field_schema = [<class 'flatland.schema.declarative.String'>, <class 'flatland.schema.declarative.String'>, <class '

class MoinMoin.items.ticket.**TicketMetaForm**(*value=Unspecified*, ***kw*)
Bases: flatland.schema.declarative.Form

field_schema = [<class 'flatland.schema.declarative.Reference'>, <class 'flatland.schema.declarative.Integer'>, <cl

class MoinMoin.items.ticket.**TicketSubmitForm**(*value=Unspecified*, ***kw*)
Bases: *MoinMoin.items.ticket.TicketForm*

field_schema = [<class 'flatland.schema.declarative.String'>, <class 'flatland.schema.declarative.String'>, <class '

submit_label = l'Submit ticket'

class MoinMoin.items.ticket.**TicketUpdateForm**(*value=Unspecified*, ***kw*)
Bases: *MoinMoin.items.ticket.TicketForm*

field_schema = [<class 'flatland.schema.declarative.String'>, <class 'flatland.schema.declarative.String'>, <class '

MoinMoin.items.ticket.**build_tree**(*comments*, *root*, *comment_tree*, *indent*)
Return an ordered list of comments related to a root comment.

Parameters

- **comments** – dict containing list of comments related to root
- **root** – a comment to the ticket description
- **comment_tree** – empty list on first call, may be populated through recursion

Return type list of comments

Returns list of tuples [comments, indent] pertaining to a comment against original description

MoinMoin.items.ticket.**check_itemid**(*self*)

MoinMoin.items.ticket.**create_comment**(*meta*, *message*)
Create a new item comment against original description, refers_to links to original.

MoinMoin.items.ticket.**file_upload**(*self*, *data_file*)

MoinMoin.items.ticket.**get_comments**(*self*)
Return a list of roots (comments to original ticket) and a dict of comments (comments to comments).

MoinMoin.items.ticket.**get_files**(*self*)

MoinMoin.items.ticket.**get_itemid_short_summary**(*rev*)

MoinMoin.items.ticket.**get_name**(*rev*)

MoinMoin.items.ticket.**message_markup**(*message*)
Add a heading with author and timestamp to message (aka ticket description).

MoinMoin.items.ticket.**render_comment_data**(*comment*)
Return a rendered comment.

Module contents

MoinMoin - high-level (frontend) items

While MoinMoin.storage cares for backend storage of items, this module cares for more high-level, frontend items, e.g. showing, editing, etc. of wiki items.

Each class in this module corresponds to an itemtype.

class MoinMoin.items.**ACLValidator**(***kw*)
Bases: flatland.validation.base.Validator

Meta Validator - currently used for validating ACLs only

acl_fail_msg = l'The ACL string is invalid'

validate(*element*, *state*)

class MoinMoin.items.**BaseChangeForm**(*value=Unspecified*, ***kw*)
Bases: *MoinMoin.security.textcha.TextChaizedForm*

field_schema = [<class 'flatland.schema.declarative.String'>, <class 'flatland.schema.declarative.String'>, <class '

submit_label = l'OK'

class MoinMoin.items.**BaseMetaForm**(*value=Unspecified*, ***kw*)
Bases: flatland.schema.declarative.Form

field_schema = [<class 'flatland.schema.declarative.String'>, <class 'flatland.schema.declarative.String'>, <class '

class MoinMoin.items.**BaseModifyForm**(*value=Unspecified*, ***kw*)
Bases: *MoinMoin.items.BaseChangeForm*

This class is abstract and only defines two factory methods; see Item._ModifyForm for the implementation.

field_schema = [<class 'flatland.schema.declarative.String'>, <class 'flatland.schema.declarative.String'>, <class '

classmethod from_item(*item*)
Construct an instance from :item.

This class method is not supposed to be overriden; subclasses should overrride the _load method instead.

classmethod from_request(*request*)
Construct an instance from :request.

Since the mapping from HTTP form (unlike from an Item instance) to Flatland Form is straightforward, there should be rarely any need to override this class method.

class MoinMoin.items.**Contentful**(*fqname*, *rev=None*, *content=None*)
Bases: *MoinMoin.items.Item*

Base class for Item subclasses that have content.

ModifyForm

class MoinMoin.items.**Default**(*fqname*, *rev=None*, *content=None*)
Bases: *MoinMoin.items.Contentful*

A "conventional" wiki item.

description = l'Wiki item'

display_name = l'Default'

do_modify()

do_show(*revid*)
Display an item. If this is not the current revision, then page content will include an H1 tag with rev-id and next-rev / prior-rev links.

doc_link(*filename*, *link_text*)
create a link to serve local doc files as help for wiki editors

itemtype = u'default'

order = -10

class MoinMoin.items.**DummyItem**(*fqname*)
Bases: object

if we have no stored Item, we use this dummy

destroy_all_revisions()

list_revisions()

class MoinMoin.items.**DummyRev**(*item*, *itemtype=None*, *contenttype=None*)
Bases: dict

if we have no stored Revision, we use this dummy

exception MoinMoin.items.**FieldNotUniqueError**
Bases: exceptions.ValueError

The Field is not a UFIELD(unique Field). Non unique fields can refer to more than one item.

class MoinMoin.items.**IndexEntry**(*relname*, *fullname*, *meta*)
Bases: tuple

fullname
Alias for field number 1

meta
Alias for field number 2

relname
Alias for field number 0

class MoinMoin.items.**Item**(*fqname*, *rev=None*, *content=None*)
Bases: object

Highlevel (not storage) Item, wraps around a storage Revision

build_index_query(*startswith=None*, *selected_groups=None*, *isglobalindex=False*)

contenttype

classmethod create(*name=u''*, *itemtype=None*, *contenttype=None*, *rev_id=u'current'*, *item=None*)
Create a highlevel Item by looking up :name or directly wrapping :item and extract the Revision designated by :rev_id revision.

The highlevel Item is created by creating an instance of Content subclass according to the item's contenttype metadata entry; The :contenttype argument can be used to override contenttype. It is used only when handling +convert (when deciding the contenttype of target item), +modify (when creating

a new item whose contenttype is not yet decided), +diff and +diffraw (to coerce the Content to a common super-contenttype of both revisions).

After that the Content instance, an instance of Item subclass is created according to the item's itemtype metadata entry, and the previously created Content instance is assigned to its content property.

delete (*comment=u''*)
delete this item (remove current name from NAME list)

description = u''

destroy (*comment=u''*, *destroy_item=False*)

display_name = u''

do_modify ()
Handle +modify requests, both GET and POST.

This method should be overridden in subclasses, providing polymorphic behavior for the +modify view.

get_index (*startswith=None*, *selected_groups=None*)

get_meta ()

get_mixed_index ()

get_prefix_match (*name*, *prefixes*)
returns the prefix match found.

get_subitem_revs ()
Create a list of subitems of this item.

Subitems are in the form of storage Revisions.

itemtype = ''

make_flat_index (*subitems*, *isglobalindex=False*)
Create two IndexEntry lists - `dirs` and `files` - from a list of subitems.

Direct subitems are added to the `files` list.

For indirect subitems, its ancestor which is a direct subitem is added to the `dirs` list. Supposing current index root is 'foo' and when 'foo/bar/la' is encountered, 'foo/bar' is added to `dirs`.

The direct subitem need not exist.

When both a subitem itself and some of its subitems are in the subitems list, it appears in both `files` and `dirs`.

Parameters **isglobalindex** – True if the query is for global indexes.

meta

meta_dict_to_text (*meta*, *use_filter=True*)
convert meta data from a dict to a text fragment

meta_filter (*meta*)
kill metadata entries that we set automatically when saving

meta_text_to_dict (*text*)
convert meta data from a text fragment to a dict

meta_to_dict (*meta*, *use_filter=True*)
convert meta data from storage object to python dict

modify (*meta*, *data*, *comment=u''*, *contenttype_guessed=None*, ***update_meta*)

name
returns the first name from the list of names.

names
returns a list of 0..n names of the item If we are dealing with a specific name (e.g field being NAME_EXACT), move it to position 0 of the list, so the upper layer can use names[0] if they want that particular name and names for the whole list. TODO make the entire code to be able to use names instead of name

order = 0

prepare_meta_for_modify (*meta*)
transform the meta dict of the current revision into a meta dict that can be used for saving next revision (after "modify").

rename (*name*, *comment=u''*)
rename this item to item <name> (replace current name by another name in the NAME list)

revert (*comment=u''*)

shown = True

subitem_prefixes
Return the possible prefixes for subitems.

class MoinMoin.items.**MixedIndexEntry** (*relname*, *fullname*, *meta*, *hassubitems*)
Bases: tuple

fullname
Alias for field number 1

hassubitems
Alias for field number 3

meta
Alias for field number 2

relname
Alias for field number 0

exception MoinMoin.items.**NameNotUniqueError**
Bases: exceptions.ValueError

An item with the same name exists.

class MoinMoin.items.**NonExistent** (*fqname*, *rev=None*, *content=None*)
Bases: *MoinMoin.items.Item*

A dummy Item for nonexistent items (when modifying, a nonexistent item with undetermined itemtype)

delete (*comment=u''*)

destroy (*comment=u''*, *destroy_item=False*)

do_modify ()

do_show (*revid*)

itemtype = u'nonexistent'

rename (*name*, *comment=u''*)

revert (*comment=u''*)

shown = False

class MoinMoin.items.**RegistryItem**
Bases: *MoinMoin.util.registry.RegistryBase*

class Entry
Bases: MoinMoin.items.Entry

RegistryItem.**register** (*e*, *shown*)
Register a factory

Parameters **factory** – Factory to register. Callable, must return an object.

class MoinMoin.items.**Userprofile**(*fqname*, *rev=None*, *content=None*)
Bases: *MoinMoin.items.Item*

Currently userprofile is implemented as a contenttype. This is a stub of an itemtype implementation of userprofile.

description = l'User profile item (not implemented yet!)'

display_name = l'User profile'

itemtype = u'userprofile'

MoinMoin.items.**get_itemtype_specific_tags**(*itemtype*)
Returns the tags of a specific itemtype

MoinMoin.items.**get_storage_revision**(*fqname*, *itemtype=None*, *contenttype=None*, *rev_id=u'current'*, *item=None*)
Get a storage Revision.

If :item is supplied it is used as the storage Item; otherwise the storage Item is looked up with :name. If it is not found (either because the item doesn't exist or the user does not have the required permissions) a DummyItem is created, and a DummyRev is created with appropriate metadata properties and the "item" property pointing to the DummyItem. The DummyRev is then returned.

If the previous step didn't end up with a DummyRev, the revision designated by :rev_id is then looked up. If it is not found, current revision is looked up and returned instead. If current revision is not found (i.e. the item has no revision), a DummyRev is created. (TODO: in the last two cases, emit warnings or throw exceptions.)

:itemtype and :contenttype are used when creating a DummyRev, where metadata is not available from the storage.

MoinMoin.items.**register**(*cls*)

MoinMoin.macro package

Submodules

MoinMoin.macro.Anchor module

MoinMoin - Anchor Macro to put an anchor at the place where it is used.

class MoinMoin.macro.Anchor.**Macro**
Bases: MoinMoin.macro._base.MacroInlineBase

macro(*content*, *arguments*, *page_url*, *alternative*)

MoinMoin.macro.Date module

MoinMoin Date macro - outputs the date for some specific point in time, adapted to the TZ settings of the user viewing the content.

class MoinMoin.macro.Date.**Macro**
Bases: *MoinMoin.macro.Date.MacroDateTimeBase*

macro(*content*, *arguments*, *page_url*, *alternative*)

class MoinMoin.macro.Date.**MacroDateTimeBase**
Bases: MoinMoin.macro._base.MacroInlineBase

parse_time(*args*)
parse a time specification argument for usage by Date and DateTime macro

Parameters **args** – YYYY-MM-DDTHH:MM:SS (plus optional Z for UTC, or +/-HHMM) or float/int UNIX timestamp

Returns UNIX timestamp (UTC)

MoinMoin.macro.DateTime module

MoinMoin DateTime macro - outputs the date and time for some specific point in time, adapted to the TZ settings of the user viewing the content.

class MoinMoin.macro.DateTime.**Macro**
Bases: *MoinMoin.macro.Date.MacroDateTimeBase*

macro(*content*, *arguments*, *page_url*, *alternative*)

MoinMoin.macro.GetText module

MoinMoin - Load I18N Text

This macro has the main purpose of supporting Help* page authors to insert the texts that a user actually sees on his screen into the description of the related features (which otherwise could get very confusing).

class MoinMoin.macro.GetText.**Macro**
Bases: MoinMoin.macro._base.MacroInlineBase

Return a translation of args, or args as is

macro(*content*, *arguments*, *page_url*, *alternative*)

MoinMoin.macro.GetVal module

MoinMoin GetVal macro - gets a value for a specified key from a dict.

class MoinMoin.macro.GetVal.**Macro**
Bases: MoinMoin.macro._base.MacroInlineBase

macro(*content*, *arguments*, *page_url*, *alternative*)

MoinMoin.macro.HighlighterList module

HighlighterList - display a list of Pygments lexers

Usage: <<HighlighterList>>

class MoinMoin.macro.HighlighterList.**Macro**
Bases: MoinMoin.macro._base.MacroBlockBase

macro(*content*, *arguments*, *page_url*, *alternative*)

MoinMoin.macro.MailTo module

MoinMoin - MailTo Macro displays an E-Mail address (either a valid mailto: link for logged in users or the obfuscated display passed as the first macro argument).

class MoinMoin.macro.MailTo.**Macro**
Bases: MoinMoin.macro._base.MacroInlineBase

macro(*content*, *arguments*, *page_url*, *alternative*)
Invocation: <<MailTo(user AT example DOT org, write me)>> where 2nd parameter is optional.

MoinMoin.macro.PagenameList module

PagenameList - list pages with names matching a string or regex

Note: PageList is a similar thing using the search engine.

class MoinMoin.macro.PagenameList.**Macro**
Bases: MoinMoin.macro._base.MacroPageLinkListBase

macro(*content*, *arguments*, *page_url*, *alternative*)

MoinMoin.macro.RandomItem module

MoinMoin - RandomItem Macro displays one or multiple random item links.

TODO: add mimetype param and only show items matching this mimetype

class MoinMoin.macro.RandomItem.**Macro**
Bases: MoinMoin.macro._base.MacroPageLinkListBase

macro(*content*, *arguments*, *page_url*, *alternative*)

MoinMoin.macro.Verbatim module

Output the input text as is: <<Verbatim(return *same* __text__ ‘’‘as’‘’ entered)>>

class MoinMoin.macro.Verbatim.**Macro**
Bases: MoinMoin.macro._base.MacroInlineBase

macro(*content*, *arguments*, *page_url*, *alternative*)

Module contents

MoinMoin - New style macros

Macros are used to implement complex and/or dynamic page content.

These new-style macros use a class interface and always work on the internal tree representation of the document.

TODO: Merge with converters

MoinMoin.mail package

Submodules

MoinMoin.mail.sendmail module

MoinMoin - email helper functions

MoinMoin.mail.sendmail.**decodeSpamSafeEmail**(*address*)
Decode obfuscated email address to standard email address

Decode a spam-safe email address in *address* by applying the following rules:

Known all-uppercase words and their translation: “DOT” -> ”.” “AT” -> “@” “DASH” -> “-“

Any unknown all-uppercase words or an uppercase letter simply get stripped. Use that to make it even harder for spam bots!

Blanks (spaces) simply get stripped.

Parameters **address** – obfuscated email address string

Return type string

Returns decoded email address

MoinMoin.mail.sendmail.**encodeAddress**(*address*, *charset*)
Encode email address to enable non ascii names

E.g. "'Jürgen Hermann" <jh@web.de>'. According to the RFC, the name part should be encoded, the address should not.

Parameters

- **address** (*unicode*) – email address, possibly using "'name" <address>' format
- **charset** (*email.Charset.Charset instance*) – specifying both the charset and the encoding, e.g quoted printable or base64.

Return type string

Returns encoded address

MoinMoin.mail.sendmail.**encodeSpamSafeEmail**(*email_address*, *obfuscation_text=''*)
Encodes a standard email address to an obfuscated address

Parameters

- **email_address** – mail address to encode. Known characters and their all-uppercase words translation:

```
"." -> " DOT "
"@" -> " AT "
"-" -> " DASH "
```

- **obfuscation_text** – optional text to obfuscate the email. All characters in the string must be alphabetic and they will be added in uppercase.

MoinMoin.mail.sendmail.**sendmail**(*subject*, *text*, *to=None*, *cc=None*, *bcc=None*, *mail_from=None*, *html=None*)
Create and send a text/plain message

Return a tuple of success or error indicator and message.

Parameters

- **subject** (*unicode*) – subject of email
- **text** (*unicode*) – email body text
- **to** (*list*) – recipients
- **cc** (*list*) – recipients (CC)
- **bcc** (*list*) – recipients (BCC)
- **mail_from** (*unicode*) – override default mail_from
- **html** (*unicode*) – html email body text

Return type tuple

Returns (is_ok, Description of error or OK message)

Module contents

MoinMoin - Package Initialization

Subpackage containing e-mail support code.

MoinMoin.script package

Subpackages

MoinMoin.script.account package

Submodules

MoinMoin.script.account.create module

MoinMoin - create a user account

class MoinMoin.script.account.create.**Create_User**(*func=None*)
Bases: flask_script.commands.Command

description = 'This command allows you to create a user account'

option_list = (<flask_script.commands.Option object>, <flask_script.commands.Option object>, <flask_script.co

run(*name*, *display_name*, *email*, *openid*, *password*)

MoinMoin.script.account.disable module

MoinMoin - disable a user account

class MoinMoin.script.account.disable.**Disable_User**(*func=None*)
Bases: flask_script.commands.Command

description = 'This command allows you to disable user accounts.'

option_list = (<flask_script.commands.Option object>, <flask_script.commands.Option object>)

run(*name*, *uid*)

MoinMoin.script.account.resetpw module

MoinMoin - set a user password

exception MoinMoin.script.account.resetpw.**Fault**
Bases: exceptions.Exception

something went wrong

exception MoinMoin.script.account.resetpw.**MailFailed**
Bases: *MoinMoin.script.account.resetpw.Fault*

raised if e-mail sending failed

exception MoinMoin.script.account.resetpw.**NoSuchUser**
Bases: *MoinMoin.script.account.resetpw.Fault*

raised if no such user exists

class MoinMoin.script.account.resetpw.**Set_Password**(*func=None*)
Bases: flask_script.commands.Command

description = 'This command allows you to set a user password.'

option_list = (<flask_script.commands.Option object>, <flask_script.commands.Option object>, <flask_script.co

run(*name*, *uid*, *password*, *all_users*, *notify*, *verbose*, *subject*, *text*, *text_file*, *skip_invalid*)

exception MoinMoin.script.account.resetpw.**UserHasNoEMail**
Bases: *MoinMoin.script.account.resetpw.Fault*

raised if user has no e-mail address in his profile

MoinMoin.script.account.resetpw.**set_password**(*uid*, *password*, *notify=False*, *skip_invalid=False*, *subject=None*, *text=None*)

Module contents

MoinMoin - User Accounts Management Scripts

MoinMoin.script.maint package

Submodules

MoinMoin.script.maint.dump_html module

MoinMoin.script.maint.index module

MoinMoin - manage whoosh indexes (building, updating, (re)moving and displaying)

class MoinMoin.script.maint.index.**IndexBuild**(*func=None*)
Bases: flask_script.commands.Command

description = 'Build the indexes.'

option_list = [<flask_script.commands.Option object>, <flask_script.commands.Option object>, <flask_script.co

run(*tmp*, *procs*, *limitmb*)

class MoinMoin.script.maint.index.**IndexCreate**(*func=None*)
Bases: flask_script.commands.Command

description = 'Create empty indexes.'

option_list = [<flask_script.commands.Option object>, <flask_script.commands.Option object>, <flask_script.co

run(*tmp*)

class MoinMoin.script.maint.index.**IndexDestroy**(*func=None*)
Bases: flask_script.commands.Command

description = 'Destroy the indexes.'

option_list = [<flask_script.commands.Option object>]

run(*tmp*)

class MoinMoin.script.maint.index.**IndexDump**(*func=None*)
Bases: flask_script.commands.Command

description = 'Dump the indexes in readable form to stdout.'

option_list = [<flask_script.commands.Option object>]

run(*tmp*)

class MoinMoin.script.maint.index.**IndexMove**(*func=None*)
Bases: flask_script.commands.Command

description = 'Move the indexes from the temporary to the normal location.'

option_list = []

run()

class MoinMoin.script.maint.index.**IndexOptimize**(*func=None*)
Bases: flask_script.commands.Command

description = 'Optimize the indexes.'

option_list = [<flask_script.commands.Option object>]

run(*tmp*)

class MoinMoin.script.maint.index.**IndexUpdate**(*func=None*)
Bases: flask_script.commands.Command

description = 'Update the indexes.'

option_list = [<flask_script.commands.Option object>]

run(*tmp*)

MoinMoin.script.maint.modify_item module

MoinMoin - get an item revision from the wiki, put it back into the wiki.

class MoinMoin.script.maint.modify_item.**GetItem**(*func=None*)
Bases: flask_script.commands.Command

description = 'Get an item revision from the wiki.'

option_list = (<flask_script.commands.Option object>, <flask_script.commands.Option object>, <flask_script.co

run(*name*, *meta_file*, *data_file*, *revid*)

class MoinMoin.script.maint.modify_item.**PutItem**(*func=None*)
Bases: flask_script.commands.Command

description = 'Put an item revision into the wiki.'

option_list = (<flask_script.commands.Option object>, <flask_script.commands.Option object>, <flask_script.co

run(*meta_file*, *data_file*, *overwrite*)

MoinMoin.script.maint.moinshell module

class MoinMoin.script.maint.moinshell.**MoinShell**(*banner=None*, *make_context=None*, *use_ipython=True*)
Bases: flask_script.commands.Command

Runs a Python shell inside Flask application context.

Parameters

- **banner** – banner appearing at top of shell when started
- **make_context** – a callable returning a dict of variables used in the shell namespace. By default returns a dict consisting of just the app.
- **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.

banner = u'"flaskg" and "app" objects are in globals now.'

description = 'Runs a Python shell inside Flask application context.'

get_context()
Returns a dict of context variables added to the shell namespace.

get_options()

run (*no_ipython*)
Runs the shell. Unless no_ipython is True or use_python is False then runs IPython shell if that is installed.

MoinMoin.script.maint.reduce_revisions module

MoinMoin - Reduce Revisions of a backend

This script removes all revisions but the last one from all selected items.

class MoinMoin.script.maint.reduce_revisions.**Reduce_Revisions** (*func=None*)
Bases: flask_script.commands.Command

description = 'This command can be used to remove all revisions but the last one from all selected items.'

option_list = (<flask_script.commands.Option object>,)

run (*query*)

MoinMoin.script.maint.serialization module

MoinMoin - backend serialization / deserialization

class MoinMoin.script.maint.serialization.**Deserialize** (*func=None*)
Bases: flask_script.commands.Command

description = 'Deserialize a file into the backend.'

option_list = [<flask_script.commands.Option object>]

run (*filename=None*)

class MoinMoin.script.maint.serialization.**Serialize** (*func=None*)
Bases: flask_script.commands.Command

description = 'Serialize the backend into a file.'

option_list = [<flask_script.commands.Option object>, <flask_script.commands.Option object>, <flask_script.co

run (*filename=None*, *backends=None*, *all_backends=False*)

MoinMoin.script.maint.serialization.**open_file** (*filename*, *mode*)

MoinMoin.script.maint.set_meta module

MoinMoin - Set Metadata of a revision

This script duplicates the last revision of the selected item and sets or removes metadata.

class MoinMoin.script.maint.set_meta.**Set_Meta** (*func=None*)
Bases: flask_script.commands.Command

description = 'This command can be used to set meta data of a new revision.'

option_list = (<flask_script.commands.Option object>, <flask_script.commands.Option object>, <flask_script.co

run (*key*, *value*, *remove*, *query*)

Module contents

MoinMoin - Maintenance Script Package

MoinMoin.script.migration package

Subpackages

MoinMoin.script.migration.moin19 package

Submodules

MoinMoin.script.migration.moin19.import19 module

MoinMoin - import content and user data from a moin 1.9 compatible storage into the moin2 storage.

TODO

- translate revno numbering into revid parents
- ACLs for attachments

class MoinMoin.script.migration.moin19.import19.**AttachmentRevision**(*item_name*, *attach_name*, *attpath*, *editlog*, *acl*)

Bases: object

moin 1.9 attachment (there is no revisioning, just 1 revision per attachment)

class MoinMoin.script.migration.moin19.import19.**EditLog**(*filename*, *buffer_size=4096*)

Bases: MoinMoin.script.migration.moin19._logfile19.LogFile

Access the edit-log and return metadata as the new api wants it.

find_attach(*attachname*)
Find metadata for some attachment name in the edit-log.

find_rev(*revno*)
Find metadata for some revno revision in the edit-log.

parser(*line*)
Parse edit-log line into fields

class MoinMoin.script.migration.moin19.import19.**ImportMoin19**(*func=None*)

Bases: flask_script.commands.Command

description = 'Import data from a moin 1.9 wiki.'

option_list = [<flask_script.commands.Option object>, <flask_script.commands.Option object>, <flask_script.co

run(*data_dir=None*)

exception MoinMoin.script.migration.moin19.import19.**KillRequested**

Bases: exceptions.Exception

raised if item killing is requested by DELETED_MODE

class MoinMoin.script.migration.moin19.import19.**PageBackend**(*path*, *deleted_mode='keep'*, *default_markup=u'wiki'*, *item_category_regex=u'(?P<all>Category(*

Bases: object

moin 1.9 page directory

class MoinMoin.script.migration.moin19.import19.**PageItem**(*backend*, *path*, *itemname*)
Bases: object

moin 1.9 page

iter_attachments()

iter_revisions()

class MoinMoin.script.migration.moin19.import19.**PageRevision**(*item*, *revno*, *path*)
Bases: object

moin 1.9 page revision

class MoinMoin.script.migration.moin19.import19.**UserBackend**(*path*)
Bases: object

moin 1.9 user directory

class MoinMoin.script.migration.moin19.import19.**UserRevision**(*path*, *uid*)
Bases: object

moin 1.9 user

migrate_subscriptions(*subscribed_items*)
Transfer subscribed_items meta to subscriptions meta

WikiFarmNames are converted to namespace names.

Parameters **subscribed_items** – a list of moin19-format subscribed_items

Returns subscriptions

MoinMoin.script.migration.moin19.import19.**hash_hexdigest**(*content*, *buf-size=4096*)

MoinMoin.script.migration.moin19.import19.**process_categories**(*meta*, *data*, *item_category_regex*)

MoinMoin.script.migration.moin19.import19.**regenerate_acl**(*acl_string*, *acl_rights_valid=['read', 'write', 'create', 'destroy', 'admin']*)

recreate ACL string to remove invalid rights

Module contents

MoinMoin - migration (upgrading) code for upgrades 1.9 -> 2.0

Module contents

MoinMoin - migration scripts

MoinMoin.script.win package

Submodules

MoinMoin.script.win.dos2unix module

Alternative for unix dos2unix utility that may be run on either windows or unix. Does not implement typical unix dos2unix command line syntax.

If passed parameter is a directory, all files in that directory are converted to unix line endings. Sub-directories are not processed. If passed parameter is a filename, only that filename is converted.

Usage: python <path_to>dos2unix.py <target_directory_or_filename>

MoinMoin.script.win.dos2unix.**convert_file**(*filename*)
Replace DOS line endings with unix line endings.

MoinMoin.script.win.wget module

Alternative for unix wget utility that may be run on either windows or unix. Does not implement typical unix wget command line syntax.

Usage: python <path_to>wget.py <url> <output_file>

Module contents

MoinMoin - Multi-platform alternatives for unix utilities

Module contents

MoinMoin - Extension Script Package

MoinMoin.script.**fatal**(*msg*)

MoinMoin.script.**main**(*default_command='moin'*, *wiki_config=None*)
console_script entry point

MoinMoin.search package

Submodules

MoinMoin.search.analyzers module

MoinMoin - Misc. tokenizers and analyzers for whoosh indexing

class MoinMoin.search.analyzers.**AclTokenizer**(*acl_rights_contents*)
Bases: whoosh.analysis.tokenizers.Tokenizer

Access control list tokenizer

class MoinMoin.search.analyzers.**MimeTokenizer**
Bases: whoosh.analysis.tokenizers.Tokenizer

Content type tokenizer

MoinMoin.search.analyzers.**item_name_analyzer**()
Analyzer behaviour:

Input: u"some item name", u"SomeItem/SubItem", u"GSOC2011"

Output: u"some", u"item", u"name"; u"Some", u"Item", u"Sub", u"Item"; u"GSOC", u"2011"

Module contents

MoinMoin - MoinMoin search package

class MoinMoin.search.**SearchForm**(*value=Unspecified*, ***kw*)
Bases: flatland.schema.declarative.Form

field_schema = [<class 'flatland.schema.declarative.String'>]

submit_label = l'Search'

validators = [<MoinMoin.search.ValidSearch object>]

class MoinMoin.search.**ValidSearch**(***kw*)
Bases: flatland.validation.base.Validator

Validator for a valid search form

too_short_query_msg = l'Search query too short.'

validate(*element*, *state*)

MoinMoin.security package

Submodules

MoinMoin.security.textcha module

MoinMoin - Text CAPTCHAs

This is just asking some (admin configured) questions and checking if the answer is as expected. It is up to the wiki admin to setup questions that a bot can not easily answer, but humans can. It is recommended to setup SITE SPECIFIC questions and not to share the questions with other sites (if everyone asks the same questions / expects the same answers, spammers could adapt to that).

TODO:

- roundtrip the question in some other way, make sure a q/a pair in the POST is for the q in the GET before
- make some nice CSS
- make similar changes to GUI editor

class MoinMoin.security.textcha.**TextCha**(*form*)
Bases: object

Text CAPTCHA support

amend_form()
Amend the form by doing the following:

•set the question if textcha is enabled, or

•make the fields optional if it isn't.

init_qa(*question=None*)
Initialize the question / answer.

Parameters **question** – If given, the given question will be used. If None, a new question will be generated.

is_enabled()
check if textchas are enabled.

They can be disabled for all languages if you use textchas = None or = {}, also they can be disabled for some specific language, like:

```
textchas = {
    'en': {
        'some question': 'some answer',
        # ...
    },
    'de': {}, # having no questions for 'de' means disabling textchas for
↪'de'
```

```
    # ...
}
```

class MoinMoin.security.textcha.**TextChaValid**(***kw*)
Bases: flatland.validation.base.Validator

Validator for TextChas

textcha_incorrect_msg = l'The entered TextCha was incorrect.'

textcha_invalid_msg = l'The TextCha question is invalid or has expired. Please try again.'

validate(*element*, *state*)

class MoinMoin.security.textcha.**TextChaizedForm**(*value=Unspecified*, ***kw*)
Bases: flatland.schema.declarative.Form

a form providing TextCha support

field_schema = [<class 'flatland.schema.declarative.String'>, <class 'flatland.schema.declarative.String'>]

MoinMoin.security.ticket module

MoinMoin - Tickets

Tickets are usually used in forms to make sure that form submissions are in response to a form the same user got from moin.

MoinMoin.security.ticket.**checkTicket**(*ticket*, ***kw*)
Check validity of a previously created ticket.

Parameters

- **ticket** – a str as created by createTicket
- **kw** – see createTicket kw

MoinMoin.security.ticket.**createTicket**(*tm=None*, ***kw*)
Create a ticket using a configured secret

Parameters

- **tm** – unix timestamp (optional, uses current time if not given)
- **kw** – key/value stuff put into ticket, must be same for ticket creation and ticket check

Module contents

MoinMoin - Wiki Security Interface and Access Control Lists

class MoinMoin.security.**ACLStringIterator**(*rights*, *aclstring*)
Bases: object

Iterator for acl string

Parse acl string and return the next entry on each call to next. Implements the Iterator protocol.

Usage:

```
iter = ACLStringIterator(rights_valid, 'user name:right')
for modifier, entries, rights in iter:
    # process data
```

next()
Return the next values from the acl string

When the iterator is finished and you try to call next, it raises a StopIteration. The iterator finishes as soon as the string is fully parsed or can not be parsed any more.

Return type 3 tuple - (modifier, [entry, ..], [right, ..])

Returns values for one item in an acl string

class MoinMoin.security.**AccessControlList**(*lines=[]*, *default=''*, *valid=None*)

Bases: *MoinMoin.util.pysupport.AutoNe*

Access Control List - controls who may do what.

Syntax of an ACL string:

[+|-]User[,User,...]:[right[,right,...]] [[+|-]SomeGroup:...] ... ... [[+|-]Known:...] [[+|-]All:...]

"User" is a user name and triggers only if the user matches. Any name can be used in acl lines, including names with spaces using exotic languages.

"SomeGroup" is a group name. The group defines its members somehow, e.g. on a wiki page of this name as first level list with the group members' names.

"Known" is a special group containing all valid / known users.

"All" is a special group containing all users (Known and Anonymous users).

"right" may be an arbitrary word like read, write or admin. Only valid words are accepted, others are ignored (see valid param). It is allowed to specify no rights, which means that no rights are given.

How ACL is processed

When some user is trying to access some ACL-protected resource, the ACLs will be processed in the order they are found. The first matching ACL will tell if the user has access to that resource or not.

For example, the following ACL tells that SomeUser is able to read and write the resources protected by that ACL, while any member of SomeGroup (besides SomeUser, if part of that group) may also admin that, and every other user is able to read it.

SomeUser:read,write SomeGroup:read,write,admin All:read

In this example, SomeUser can read and write but can not admin items. Rights that are NOT specified on the right list are automatically set to NO.

Using Prefixes

To make the system more flexible, there are also two modifiers: the prefixes "+" and "-".

+SomeUser:read -OtherUser:write

The acl line above will grant SomeUser read right, and deny OtherUser write right, but will NOT block automatically all other rights for these users. For example, if SomeUser asks to write, the above acl line does not define if he can or can not write. He will be able to write if the acls checked before or afterwards allow this (see configuration options).

Using prefixes, this acl line:

SomeUser:read,write SomeGroup:read,write,admin All:read

Can be written as:

-SomeUser:admin SomeGroup:read,write,admin All:read

Or even:

+All:read -SomeUser:admin SomeGroup:read,write,admin

Note that you probably would not want to use the second and third examples in ACL entries of some item. They are very useful in the wiki configuration though.

has_acl()
Checks whether we have a real acl here.

may(*name*, *dowhat*)
May <name> <dowhat>? Returns boolean answer.

Note: this just checks THIS ACL, the before/default/after ACL must be handled elsewhere, if needed.

special_users = ['All', 'Known', 'Trusted']

class MoinMoin.security.**DefaultSecurityPolicy**(*user*)
Bases: object

Basic interface for user permissions and system policy.

If you want to define your own policy, inherit from DefaultSecurityPolicy, so that when new permissions are defined later, you will inherit their default behaviour.

Then assign your new class (not an instance!) to "SecurityPolicy" in the wiki configuration.

When subclassing this class, you must extend the class methods, not replace them, or you might break the ACLs in the wiki.

Correct subclassing looks like this:

```
class MySecPol(DefaultSecurityPolicy):
    def read(self, itemname):
        # Your special security rule
        if something:
            return False

        # Do not just return True or you break (ignore) ACLs!
        # This call will return correct permissions by checking ACLs:
        return super(MySecPol, self).read(itemname)
```

read(*itemname*)
read permission is special as we have 2 kinds of read capabilities:

•READ - gives permission to read, unconditionally

•PUBREAD - gives permission to read, when published

MoinMoin.security.**require_permission**(*permission*)
view decorator to require a specific permission

if the permission is not granted, abort with 403

MoinMoin.signalling package

Submodules

MoinMoin.signalling.log module

MoinMoin - logging signal handlers

MoinMoin.signalling.log.**log_exception**(*sender*, *exception*, ***extra*)

MoinMoin.signalling.log.**log_item_displayed**(*app*, *fqname*)

MoinMoin.signalling.log.**log_item_modified**(*app*, *fqname*, ***kwargs*)

MoinMoin.signalling.signals module

MoinMoin - signals

We define all signals here to avoid typos/conflicts in the signal name.

Module contents

MoinMoin - signalling support

MoinMoin uses blinker for sending signals and letting listeners subscribe to signals.

MoinMoin.storage package

Subpackages

MoinMoin.storage.backends package

Submodules

MoinMoin.storage.backends.fileserver module

MoinMoin - fileserver backend, exposing part of the filesystem (read-only)

Files show as single revision items.

- metadata is made up from fs metadata + mimetype guessing
- data is read from the file

Directories create a virtual directory item, listing the files in that directory.

class MoinMoin.storage.backends.fileserver.**Backend**(*path*)
Bases: *MoinMoin.storage.backends.BackendBase*

exposes part of the filesystem (read-only)

close()

classmethod from_uri(*uri*)

open()

retrieve(*key*)

MoinMoin.storage.backends.stores module

MoinMoin - backend that ties together 2 key/value stores

A meta store (a ByteStore):

- key = revid UUID (bytes, ascii)
- value = bytes (bytes, utf-8)

A data store (a FileStore):

- key = dataid UUID (bytes, ascii)
- value = file (gets/returns open file instances, to read/write binary data)

See the stores package for already implemented key/value stores.

class MoinMoin.storage.backends.stores.**Backend**(*meta_store*, *data_store*)
Bases: *MoinMoin.storage.backends.BackendBase*

ties together a store for metadata and a store for data, readonly

close()

classmethod from_uri(*uri*)

open()

retrieve(*metaid*)

class MoinMoin.storage.backends.stores.**MutableBackend**(*meta_store*, *data_store*)
Bases: *MoinMoin.storage.backends.stores.Backend*, *MoinMoin.storage.backends.MutableBackendBase*

same as Backend, but read/write

create()

destroy()

remove(*metaid*, *destroy_data*)

store(*meta*, *data*)

Module contents

MoinMoin - backend base classes

class MoinMoin.storage.backends.**BackendBase**
Bases: object

ties together a store for metadata and a store for data, readonly

close()
close the backend, free resources (except the stored meta/data!)

classmethod from_uri(*uri*)
create an instance using the data given in uri

open()
open the backend, allocate resources

retrieve(*metaid*)
return meta, data related to metaid

class MoinMoin.storage.backends.**MutableBackendBase**
Bases: *MoinMoin.storage.backends.BackendBase*

same as Backend, but read/write

create()
create the backend

destroy()
destroy the backend, erase all meta/data it contains

remove(*metaid*)
delete meta, data related to metaid from the backend

store(*meta*, *data*)
store meta, data into the backend, return the metaid

MoinMoin.storage.middleware package

Submodules

MoinMoin.storage.middleware.indexing module

MoinMoin - indexing middleware

The backends and stores moin uses are rather simple, it is mostly just a unsorted / unordered bunch of revisions (meta and data) with iteration.

The indexer middleware adds the needed power: after all metadata and data is indexed, we can do all sorts of operations on the indexer level: * searching * lookup by name, uuid, ... * selecting * listing

Using Whoosh (a fast pure-Python indexing and search library), we build, maintain and use 2 indexes:

- "all revisions" index (big, needed for history search)
- "latest revisions" index (smaller, just the current revisions)

When creating or destroying revisions, indexes are automatically updated.

There is also code to do a full index rebuild in case it gets damaged, lost or needs rebuilding for other reasons. There is also index update code to do a quick "intelligent" update of a "mostly ok" index, that just adds, updates, deletes stuff that is different in backend compared to current index.

Indexing is the only layer that can easily deal with **names** (it can easily translate names to UUIDs and vice versa) and with **items** (it knows current revision, it can easily list and order historial revisions), using the index.

The layers below are using UUIDs to identify revisions meta and data:

- revid (metaid) - a UUID identifying a specific revision (revision metadata)
- dataid - a UUID identifying some specific revision data (optional), it is just stored into revision metadata.
- itemid - a UUID identifying an item (== a set of revisions), it is just stored into revision metadata. itemid is only easily usable on indexing level.

Many methods provided by the indexing middleware will be fast, because they will not access the layers below (like the backend), but just the index files, usually it is even just the small and thus quick latest-revs index.

class MoinMoin.storage.middleware.indexing.**IndexingMiddleware**(*index_storage*, *backend*, *wiki_name=None*, *acl_rights_contents=[]*, ***kw*)

Bases: object

close()
Close all indexes.

create(*tmp=False*)
Create all indexes (empty).

create_item(***query*)
Return item identified by the query (must be a new item).

Kwargs query e.g. name_exact=u"Foo" or itemid="..." or ... (must be a unique field-name=value for the latest-revs index)

destroy(*tmp=False*)
Destroy all indexes.

document(*idx_name='latest_revs'*, ***kw*)
Return a Revision matching the kw args.

documents(*idx_name='latest_revs'*, ***kw*)
Yield Revisions matching the kw args.

dump(*tmp=False*, *idx_name='latest_revs'*)
Yield key/value tuple lists for all documents in the indexes, fields sorted.

existing_item(***query*)
Return item identified by query (must be an existing item).

Kwargs query e.g. name_exact=u"Foo" or itemid="..." or ... (must be a unique field-name=value for the latest-revs index)

get_item(***query*)
Return item identified by the query (may be a new or existing item).

Kwargs query e.g. name_exact=u"Foo" or itemid="..." or ... (must be a unique field-name=value for the latest-revs index)

get_storage(*tmp=False*, *create=False*)
Get the whoosh storage (whoosh supports different kinds of storage, e.g. to filesystem or to GAE). Currently we only support the FileStorage.

get_storage_params(*tmp=False*)

has_item(*name*)

index_revision(*meta*, *content*, *backend_name*, *async=False*)
Index a single revision, add it to all-revs and latest-revs index.

Parameters

- **meta** – metadata dict
- **content** – preprocessed (filtered) indexable content
- **async** – if True, use the AsyncWriter, otherwise use normal writer

move_index()
Move freshly built indexes from tmp storage to normal storage

open()
Open all indexes.

optimize_backend()
Optimize backend / collect garbage to safe space:

•deleted items: destroy them? use a deleted_max_age?

•user profiles: only keep latest revision?

•normal wiki items: keep by max_revisions_count / max_age

•deduplicate data (determine dataids with same hash, fix references to point to one of them)

•remove unreferenced dataids (destroyed revisions, deduplicated stuff)

optimize_index(*tmp=False*)
Optimize whoosh index.

query_parser(*default_fields*, *idx_name='latest_revs'*)
Build a query parser for a list of default fields.

rebuild(*tmp=False*, *procs=1*, *limitmb=256*)
Add all items/revisions from the backends of this wiki to the index (which is expected to have no items/revisions from this wiki yet).

Note: index might be shared by multiple wikis, so it is: create, rebuild wiki1, rebuild wiki2, ... create (tmp), rebuild wiki1, rebuild wiki2, ..., move

remove_revision(*revid*, *async=True*)
Remove a single revision from indexes.

search(*q*, *idx_name='latest_revs'*, ***kw*)
Search with query q, yield Revisions.

search_page(*q*, *idx_name='latest_revs'*, *pagenum=1*, *pagelen=10*, ***kw*)
Same as search, but with paging support.

update(*tmp=False*)
Make sure index reflects current backend state, add missing stuff, remove outdated stuff.

This is intended to be used: * after a full rebuild that was done at tmp location * after wiki is made read-only or taken offline * after the index was moved to the normal index location

Reason: new revisions that were created after the rebuild started might be missing in new index.

Returns index changed (bool)

class MoinMoin.storage.middleware.indexing.**Item**(*indexer*, *latest_doc=None*, ***query*)
Bases: *MoinMoin.storage.middleware.indexing.PropertiesMixin*

classmethod create(*indexer*, ***query*)
Create a new item and return it, raise exception if it already exists.

destroy_all_revisions()
Destroy all revisions of this item.

destroy_revision(*revid*)
Destroy revision <revid>.

classmethod existing(*indexer*, ***query*)
Get an existing item and return it, raise exception if it does not exist.

get_revision(*revid*, *doc=None*)
Similar to item[revid], but you can optionally give an already existing whoosh result document for the given revid to avoid backend accesses for some use cases.

itemid

iter_revs()
Iterate over Revisions belonging to this item.

meta

parentids
compute list of parent itemids

Returns parent itemids (set)

preprocess(*meta*, *data*)
preprocess a revision before it gets stored and put into index.

store_all_revisions(*meta*, *data*)
Store over all revisions of this item.

store_revision(*meta*, *data*, *overwrite=False*, *trusted=False*, *name=None*, *action=u'SAVE'*, *remote_addr=None*, *userid=None*, *wikiname=None*, *contenttype_current=None*, *contenttype_guessed=None*, *acl_parent=None*, *return_rev=False*, *fqname=None*)
Store a revision into the backend, write metadata and data to it.

Usually this will be a new revision, either of an existing item or a new item. With overwrite mode, we can also store over existing revisions.

Parameters

- **overwrite** – if True, allow overwriting of existing revs.
- **return_rev** – if True, return a Revision instance of the just created revision

Returns a Revision instance or None

class MoinMoin.storage.middleware.indexing.**Meta**(*revision*, *doc*, *meta=None*)
Bases: _abcoll.Mapping

class MoinMoin.storage.middleware.indexing.**PropertiesMixin**
Bases: object

PropertiesMixin offers methods to find out some additional information from meta.

acl

fqname
return the fully qualified name including the namespace: NS:NAME

fqnames
return the fully qualified names including the namespace: NS:NAME

fqparentnames
return the fully qualified parent names including the namespace: NS:NAME

mtime

name

names

namespace

parentnames
compute list of parent names (same order as in names, but no dupes)

Returns parent names (list of unicode)

ptime

class MoinMoin.storage.middleware.indexing.**Revision**(*item*, *revid*, *doc=None*, *name=None*)
Bases: *MoinMoin.storage.middleware.indexing.PropertiesMixin*

An existing revision (exists in the backend).

close()

data

set_context(*context*)

MoinMoin.storage.middleware.indexing.**backend_subscriptions_to_index**(*subscriptions*)
Split subscriptions list to subscription_ids and subscription_patterns lists which match the fields of the whoosh schema

Parameters **subscriptions** – user subscriptions meta

Returns tuple containing a list of subscription_ids and a list of subscription_patterns

MoinMoin.storage.middleware.indexing.**backend_to_index**(*meta*, *content*, *schema*, *wikiname*, *backend_name*)
Convert backend metadata/data to a whoosh document.

Parameters

- **meta** – revision meta from moin backend
- **content** – revision data converted to indexable content
- **schema** – whoosh schema
- **wikiname** – interwikiname of this wiki

Returns document to put into whoosh index

MoinMoin.storage.middleware.indexing.**convert_to_indexable**(*meta*, *data*, *item_name=None*, *is_new=False*)
Convert revision data to a indexable content.

Parameters

- **meta** – revision metadata (gets updated as a side effect)
- **data** – revision data (file-like) please make sure that the content file is ready to read all indexable content from it. if you have just written that content or already read from it, you need to call rev.seek(0) before calling convert_to_indexable(rev).
- **is_new** – if this is for a new revision and we shall modify metadata as a side effect

Returns indexable content, text/plain, unicode object

MoinMoin.storage.middleware.indexing.**get_names**(*meta*)

Get the (list of) names from meta data and deal with misc. bad things that can happen then (while not all code is fixed to do it correctly).

TODO make sure meta[NAME] is always a list of unicode

Parameters **meta** – a metadata dictionary that might have a NAME key

Returns list of names

MoinMoin.storage.middleware.protecting module

MoinMoin - protecting middleware

This checks ACLs (access control lists), so a user will not be able to do operations without the respective permissions.

Note: for method / attribute docs, please see the same methods / attributes in IndexingMiddleware class.

exception MoinMoin.storage.middleware.protecting.**AccessDenied**

Bases: exceptions.Exception

raised when a user is denied access to an Item or Revision by ACL.

class MoinMoin.storage.middleware.protecting.**ProtectedItem**(*protector*, *item*)

Bases: object

acl

allows(*right*, *user_names=None*)

Check if usernames may have <right> access on this item.

Parameters

- **right** – the right to check
- **user_names** – user names to use for permissions check (default is to use the user names doing the current request)

Return type bool

Returns True if you have permission or False

destroy_all_revisions()

destroy_revision(*revid*)

fqname

fqnames

full_acls()

iterator over all alternatively possible full acls for this item, including before/default/after acl.

get_revision(*revid*)

itemid

iter_revs()

name

parentids

parentnames

require(**capabilities*)
require that at least one of the capabilities is allowed

store_all_revisions(*meta*, *data*)

store_revision(*meta*, *data*, *overwrite=False*, *return_rev=False*, *fqname=None*, ***kw*)

class MoinMoin.storage.middleware.protecting.**ProtectedRevision**(*protector*, *rev*, *p_item=None*)

Bases: object

allows(*capability*)

close()

data

fqname

fqnames

meta

name

require(**capabilities*)
require that at least one of the capabilities is allowed

revid

set_context(*context*)

class MoinMoin.storage.middleware.protecting.**ProtectingMiddleware**(*indexer*, *user*, *acl_mapping*)

Bases: object

create_item(***query*)

document(*idx_name='latest_revs'*, ***kw*)

documents(*idx_name='latest_revs'*, ***kw*)

existing_item(***query*)

get_item(***query*)

has_item(*name*)

may(*fqname*, *capability*, *usernames=None*)

query_parser(*default_fields*, *idx_name='latest_revs'*)

search(*q*, *idx_name='latest_revs'*, ***kw*)

search_page(*q*, *idx_name='latest_revs'*, *pagenum=1*, *pagelen=10*, ***kw*)

MoinMoin.storage.middleware.protecting.**pchecker**(*right*, *allowed*, *item*)
some permissions need additional checking

MoinMoin.storage.middleware.routing module

MoinMoin - namespaces middleware

Routes requests to different backends depending on the namespace.

class MoinMoin.storage.middleware.routing.**Backend**(*namespaces*, *backends*)
Bases: *MoinMoin.storage.backends.MutableBackendBase*

namespace dispatcher, behaves readonly for readonly mounts

close()

create()

destroy()

open()

remove(*backend_name*, *revid*, *destroy_data*)

retrieve(*backend_name*, *revid*)

store(*meta*, *data*)

MoinMoin.storage.middleware.serialization module

MoinMoin - backend serialization / deserialization

We use a simple custom format here:

```
4 bytes length of meta (m)
m bytes metadata (json serialization, utf-8 encoded)
        (the metadata contains the data length d in meta[SIZE])
d bytes binary data
... (repeat for all meta/data)
4 bytes 00 (== length of next meta -> there is none, this is the end)
```

MoinMoin.storage.middleware.serialization.**deserialize**(*src*, *backend*)

MoinMoin.storage.middleware.serialization.**serialize**(*backend*, *dst*)

MoinMoin.storage.middleware.serialization.**serialize_iter**(*backend*)

MoinMoin.storage.middleware.serialization.**serialize_rev**(*meta*, *data*)

MoinMoin.storage.middleware.validation module

MoinMoin - validation for storage meta / data

validation modes:

trusted == False: for metadata coming from user input (like from web form) - in this mode some values will be forced (e.g. mtime, address, hostname, ...).

trusted == True: for metadata coming from trusted sources (like loading backups, tests, ...)

The mode trusted=True/False and the values for forcing can be given as extra params to store_revision (see indexing module).

If supplied metadata is missing some values that are required and have sane defaults, the validators may implant the defaults into the metadata or reject the data.

MoinMoin.storage.middleware.validation.**ContentMetaSchema**
alias of *DuckDict*

class MoinMoin.storage.middleware.validation.**DuckDict**(*value=Unspecified*, ***kw*)
Bases: flatland.schema.containers.Dict

policy = 'duck'

MoinMoin.storage.middleware.validation.**UserMetaSchema**
alias of *DuckDict*

MoinMoin.storage.middleware.validation.**acl_validator**(*element*, *state*)
an acl, also checks if changing acl is allowed

MoinMoin.storage.middleware.validation.**action_validator**(*element*, *state*)
an action

MoinMoin.storage.middleware.validation.**address_validator**(*element*, *state*)
an IP address

MoinMoin.storage.middleware.validation.**comment_validator**(*element*, *state*)
a comment

MoinMoin.storage.middleware.validation.**contenttype_validator**(*element*, *state*)
a supported content type

MoinMoin.storage.middleware.validation.**hash_validator**(*element*, *state*)
a content hash

MoinMoin.storage.middleware.validation.**hostname_validator**(*element*, *state*)
a hostname (dns name)

MoinMoin.storage.middleware.validation.**itemid_validator**(*element*, *state*)
an itemid is a uuid that identifies an item

MoinMoin.storage.middleware.validation.**mtime_validator**(*element*, *state*)
a modification time (UNIX timestamp)

MoinMoin.storage.middleware.validation.**name_validator**(*element*, *state*)
a (item/revision) name

MoinMoin.storage.middleware.validation.**namespace_validator**(*element*, *state*)
a namespace (part of a wiki site)

MoinMoin.storage.middleware.validation.**revid_validator**(*element*, *state*)
a revid is a uuid that identifies a revision

MoinMoin.storage.middleware.validation.**size_validator**(*element*, *state*)
a content size

MoinMoin.storage.middleware.validation.**subscription_validator**(*element*, *state*)
a subscription

MoinMoin.storage.middleware.validation.**tag_validator**(*element*, *state*)
a tag

MoinMoin.storage.middleware.validation.**user_contenttype_validator**(*element*, *state*)
user profile content type

MoinMoin.storage.middleware.validation.**userid_validator**(*element*, *state*)
a userid is a uuid that identifies a user (profile)

MoinMoin.storage.middleware.validation.**uuid_validator**(*element*, *state*)
a uuid must be a hex unicode string of specific length

MoinMoin.storage.middleware.validation.**validate_data**(*meta*, *data*)
validate the data contents, if possible

Parameters

- **meta** – metadata dict
- **data** – data file

Returns validation ok [bool]

MoinMoin.storage.middleware.validation.**wikiname_validator**(*element*, *state*)
a wikiname (name of the wiki site)

Module contents

MoinMoin - misc. middleware

Middleware sits either on a backend or on another middleware.

MoinMoin.storage.stores package

Submodules

MoinMoin.storage.stores.fs module

MoinMoin - filesystem store

Store into filesystem, one file per k/v pair.

class MoinMoin.storage.stores.fs.**BytesStore**(*path*)
Bases: *MoinMoin.storage.stores.BytesMutableStoreMixin*, *MoinMoin.storage.stores.fs.FileStore*, *MoinMoin.storage.stores.BytesMutableStoreBase*

filesystem BytesStore

class MoinMoin.storage.stores.fs.**FileStore**(*path*)
Bases: *MoinMoin.storage.stores.FileMutableStoreBase*

A simple filesystem-based store.

keys are required to be valid filenames.

create()

destroy()

classmethod from_uri(*uri*)

MoinMoin.storage.stores.kc module

MoinMoin.storage.stores.kt module

MoinMoin - kyoto tycoon store

Stores k/v pairs into a Kyoto Tycoon server. Kyoto Tycoon is a network server for kyoto cabinet, remote or multi-process usage is possible).

class MoinMoin.storage.stores.kt.**BytesStore**(*host='127.0.0.1'*, *port=1978*, *timeout=30*)
Bases: MoinMoin.storage.stores.kt._Store, *MoinMoin.storage.stores.BytesMutableStoreBase*

get(*key*)

set(*key*, *value*, *xt=None*)

class MoinMoin.storage.stores.kt.**FileStore**(*host='127.0.0.1'*, *port=1978*, *timeout=30*)
Bases: MoinMoin.storage.stores.kt._Store, *MoinMoin.storage.stores.FileMutableStoreBase*

get(*key*)

set(*key*, *value*, *xt=None*)

MoinMoin.storage.stores.memory module

MoinMoin - memory store

Stores k/v pairs into memory (RAM, non-persistent!).

Note: likely this is mostly useful for unit tests.

class MoinMoin.storage.stores.memory.**BytesStore**
Bases: *MoinMoin.storage.stores.BytesMutableStoreBase*

A simple dict-based in-memory store. No persistence!

close()

create()

destroy()

classmethod from_uri(*uri*)

open()

class MoinMoin.storage.stores.memory.**FileStore**
Bases: *MoinMoin.storage.stores.FileMutableStoreMixin*, *MoinMoin.storage.stores.memory.BytesStore*, *MoinMoin.storage.stores.FileMutableStoreBase*

memory FileStore

MoinMoin.storage.stores.mongodb module

MoinMoin.storage.stores.sqla module

MoinMoin.storage.stores.sqlite module

MoinMoin - sqlite3 key/value store

This store stores into sqlite3 table, using a single db file in the filesystem. You can use the same db file for multiple stores, just using a different table name.

Optionally, you can use zlib/"gzip" compression.

class MoinMoin.storage.stores.sqlite.**BytesStore**(*db_name*, *table_name='store'*, *compression_level=0*)
Bases: *MoinMoin.storage.stores.BytesMutableStoreBase*

A simple sqlite3 based store.

close()

create()

destroy()

classmethod from_uri(*uri*)
Create a new cls instance using the parameters provided in the uri

Parameters

- **cls** – Class to create
- **uri** – The URI should follow the following template db_name::table_name::compression_level where table_name and compression level are optional

open()

class MoinMoin.storage.stores.sqlite.**FileStore**(*db_name*, *table_name='store'*, *compression_level=0*)

Bases: *MoinMoin.storage.stores.FileMutableStoreMixin*, *MoinMoin.storage.stores.sqlite.BytesStore*, *MoinMoin.storage.stores.FileMutableStoreBase*

sqlite FileStore

MoinMoin.storage.stores.wrappers module

MoinMoin - store wrappers

class MoinMoin.storage.stores.wrappers.**ByteToStreamWrappingStore**(*stream_store*)

Bases: _abcoll.MutableMapping

Module contents

MoinMoin - simple key/value stores.

If some kvstore implementation you'ld like to use is missing from this package, you can likely implement it adding very little and rather easy code.

class MoinMoin.storage.stores.**BytesMutableStoreBase**(**kw*)

Bases: *MoinMoin.storage.stores.MutableStoreBase*

class MoinMoin.storage.stores.**BytesMutableStoreMixin**

Bases: object

mix this into a FileMutableStore to get a BytesMutableStore, like shown here:

class BytesStore(BytesMutableStoreMixin, FileStore, BytesMutableStoreBase): # that's all, nothing more needed

class MoinMoin.storage.stores.**BytesStoreBase**(**kw*)

Bases: *MoinMoin.storage.stores.StoreBase*

class MoinMoin.storage.stores.**FileMutableStoreBase**(**kw*)

Bases: *MoinMoin.storage.stores.MutableStoreBase*

class MoinMoin.storage.stores.**FileMutableStoreMixin**

Bases: object

mix this into a BytesMutableStore to get a FileMutableStore, like shown here:

class FileStore(FileMutableStoreMixin, BytesStore, FileMutableStoreBase) # that's all, nothing more needed

class MoinMoin.storage.stores.**FileStoreBase**(**kw*)

Bases: *MoinMoin.storage.stores.StoreBase*

class MoinMoin.storage.stores.**MutableStoreBase**(**kw*)

Bases: *MoinMoin.storage.stores.StoreBase*, _abcoll.MutableMapping

A simple read/write key/value store.

create()

create an empty store

destroy()

destroy the store (erase all stored data, remove store)

class MoinMoin.storage.stores.**StoreBase**(**kw*)

Bases: _abcoll.Mapping

A simple read-only key/value store.

close()
close the store, stop using it, free resources (except stored data)

classmethod from_uri(*uri*)
return an instance constructed from the given uri

open()
open the store, prepare it for usage

Submodules

MoinMoin.storage.error module

MoinMoin storage errors

exception MoinMoin.storage.error.**BackendError**(*message*)
Bases: *MoinMoin.storage.error.StorageError*

Raised if the backend couldn't commit the action.

exception MoinMoin.storage.error.**ItemAlreadyExistsError**(*message*)
Bases: *MoinMoin.storage.error.BackendError*

Raised if the Item you are trying to create already exists.

exception MoinMoin.storage.error.**NoSuchItemError**(*message*)
Bases: *MoinMoin.storage.error.BackendError*

Raised if the requested item does not exist.

exception MoinMoin.storage.error.**NoSuchRevisionError**(*message*)
Bases: *MoinMoin.storage.error.BackendError*

Raised if the requested revision of an item does not exist.

exception MoinMoin.storage.error.**RevisionAlreadyExistsError**(*message*)
Bases: *MoinMoin.storage.error.BackendError*

Raised if the Revision you are trying to create already exists.

exception MoinMoin.storage.error.**StorageError**(*message*)
Bases: *MoinMoin.error.CompositeError*

General class for exceptions on the storage layer.

Module contents

MoinMoin - storage subsystem

We use a layered approach like this:

```
Indexing Middleware               does complex stuff like indexing, searching,
|                                 listing, lookup by name, ACL checks, ...
v
Routing  Middleware               dispatches to multiple backends based on the
|                |                namespace
v                v
"stores" Backend  Other Backend   simple stuff: store, get, destroy revisions
|           |
v           v
meta store  data store            simplest stuff: store, get, destroy and iterate
                                  over key/value pairs
```

MoinMoin.storage.**backend_from_uri**(*uri*)
create a backend instance for uri

MoinMoin.storage.**create_mapping**(*uri*, *namespaces*, *backends*, *acls*)

MoinMoin.storage.**create_simple_mapping**(*uri='stores:fs:instance'*, *default_acl=None*, *userprofiles_acl=None*)
When configuring storage, the admin needs to provide a namespace_mapping. To ease creation of such a mapping, this function provides sane defaults for different types of stores. The admin can just call this function, pass a hint on what type of stores he wants to use and a proper mapping is returned.

Params uri '<backend_name>:<backend_uri>' (general form) backend_name must be a backend module name (e.g. stores) the backend_uri must have a %(backend)s placeholder, it gets replaced by the name of the backend (a simple, ascii string) and result is given to to that backend's constructor

for the 'stores' backend, backend_uri looks like '<store_name>:<store_uri>' store_name must be a store module name (e.g. fs) the store_uri must have a %(kind)s placeholder, it gets replaced by 'meta' or 'data' and the result is given to that store's constructor

e.g.: 'stores:fs:/path/to/store/%(backend)s/%(kind)s' will create a mapping using the 'stores' backend with 'fs' stores and everything will be stored to below /path/to/store/.

MoinMoin.themes package

Module contents

MoinMoin - Theme Support

class MoinMoin.themes.**ThemeSupport**(*cfg*)
Bases: object

Support code for template feeding.

get_action_tabs(*fqname*, *current_endpoint*)
Create a list of commonly used item views. Used by Basic theme.

Return type *list*

Returns list of item views

get_endpoint_iconmap()

get_fqnames(*fqname*)
Return the list of other fqnames associated with the item.

get_local_panel(*fqname*)
Split uncommonly used cfg.item views into user actions, item actions, and view options.

Return type *list*

Returns list of lists containing: user actions, item actions, and view options for Basic theme

get_namespaces(*ns=None*)
Return the list of tuples (composite name, namespace) referring to namespaces other than the current namespace.

item_exists(*itemname*)
Check whether the item pointed to by the given itemname exists or not

Return type boolean

Returns whether item pointed to by the link exists or not

location_breadcrumbs(*fqname*)
Assemble the location using breadcrumbs (was: title)

Return type *list*

Returns location breadcrumbs items in tuple (segment_name, fq_name, exists)

login_url()
Return URL usable for user login

Return type unicode (or None, if no login url is supported)

Returns url for user login

navibar(**args*, ***kw*)
Assemble the navibar

Return type *list*

Returns list of tuples (css_class, url, link_text, title)

parent_item(*item_name*)
Return name of parent item for the current item

Return type unicode

Returns parent item name

path_breadcrumbs()
Assemble the path breadcrumbs (a.k.a.: trail)

Return type *list*

Returns path breadcrumbs items in tuple (wiki_name, item_name, url, exists, err)

split_navilink(*text*)
Split navibar links into pagename, link to page

Admin or user might want to use shorter navibar items by using the [[page|title]] or [[url|title]] syntax.

Supported syntax:

- PageName
- WikiName:PageName
- wiki:WikiName:PageName
- url
- all targets as seen above with title: [[target|title]]

Parameters **text** – the text used in config or user preferences

Return type tuple

Returns pagename or url, link to page or url

subitem_index(*fqname*)
Get a list of subitems for the given fqname

Return type *list*

Returns list of item tuples (item_name, item_title, item_mime_type, has_children)

userhome()
Assemble arguments used to build user homepage link

Return type tuple

Returns arguments of user homepage link in tuple (wiki_href, display_name, title, exists)

MoinMoin.themes.**contenttype_to_class**(*contenttype*)
Convert a contenttype string to a css class.

MoinMoin.themes.**get_assigned_to_info**(*meta*)

MoinMoin.themes.**get_current_theme**()

MoinMoin.themes.**get_editor_info**(*meta*, *external=False*)
Create a dict of formatted user info.

Return type dict

Returns dict of formatted user info such as name, ip addr, email,...

MoinMoin.themes.**render_template**(*template*, ***context*)

MoinMoin.themes.**setup_jinja_env**()

MoinMoin.themes.**shorten_ctype**(*contenttype*)
Returns user understandable terms for contenttype.

Parameters **contenttype** – contains the long form of the contenttype

Return type unicode

Returns user understandable version of contenttype

MoinMoin.themes.**shorten_fqname**(*fqname*, *length=25*)
Shorten fqname

Shorten a given long fqname so that it looks good depending upon whether the field is a UUID or not.

Parameters

- **fqname** – fqname, namedtuple
- **length** – maximum length for shortened fqnames in case the field is not a UUID.

Return type unicode

Returns shortened fqname.

MoinMoin.themes.**shorten_id**(*name*, *length=7*)
Shorten IDs to specified length

Shorten long IDs into just the first <length> characters. There's no need to display the whole IDs everywhere.

Parameters

- **name** – item name, unicode
- **length** – Maximum length of the resulting ID, int

Return type unicode

Returns <name> truncated to <length> characters

MoinMoin.themes.**shorten_item_name**(*name*, *length=25*)
Shorten item names

Shorten very long item names that tend to break the user interface. The short name is usually fine, unless really stupid long names are used (WYGIWYD).

Parameters

- **name** – item name, unicode
- **length** – maximum length for shortened item names, int

Return type unicode

Returns shortened version.

MoinMoin.themes.**themed_error**(*e*)

MoinMoin.themes.**time_hh_mm**(*dt*)
Convert a datetime object into a short string of the form HH:MM where HH varies from 0 to 23.

MoinMoin.themes.**utctimestamp**(*dt*)
convert a datetime object (UTC) to a UNIX timestamp (UTC)

Note: time library writers seem to have a distorted relationship to inverse functions and also to UTC (see time.gmtime, see datetime.utcfromtimestamp).

MoinMoin.util package

Submodules

MoinMoin.util.StringIOClosing module

class MoinMoin.util.StringIOClosing.**StringIO**(*buf=''*)
Bases: StringIO.StringIO

same as StringIO from stdlib, but enhanced with a context manager, so it can be used within a "with" statement and gets automatically closed when the with-block is left. The standard "file" object behaves that way, so a StringIO "file emulation" should behave the same.

MoinMoin.util.SubProcess module

Enhanced subprocess.Popen subclass, supporting:

- .communicate() with timeout

Sample usage: out, err = Popen(...).communicate(input, timeout=300)

class MoinMoin.util.SubProcess.**Popen**(*args*, *bufsize=0*, *executable=None*, *stdin=None*, *stdout=None*, *stderr=None*, *preexec_fn=None*, *close_fds=False*, *shell=False*, *cwd=None*, *env=None*, *universal_newlines=False*, *startupinfo=None*, *creationflags=0*)
Bases: subprocess.Popen

communicate(*input=None*, *timeout=None*)
Interact with process: Send data to stdin. Read data from stdout and stderr, until end-of-file is reached. Wait for process to terminate. The optional input argument should be a string to be sent to the child process, or None, if no data should be sent to the child.

communicate() returns a tuple (stdout, stderr).

MoinMoin.util.SubProcess.**exec_cmd**(*cmd*, *input=None*, *timeout=None*)

MoinMoin.util.clock module

MoinMoin - Clock

class MoinMoin.util.clock.**Clock**
Bases: object

Helper class for measuring the time needed to run code.

Usage: flaskg.clock.start('mytimer') # do something flaskg.clock.stop('mytimer') # or if you want to use its value later timerval = flaskg.clock.stop('mytimer')

Starting a timer multiple times is supported but the one started last has to be stopped first.

start(*timer*)

stop(*timer*)

MoinMoin.util.clock.**add_timing**(*f*, *name=None*)

MoinMoin.util.clock.**timed**(*name=None*)

MoinMoin.util.crypto module

MoinMoin - Cryptographic and random functions

Features:

- generate password recovery tokens
- verify password recovery tokens
- generate random strings of given length (for salting)

MoinMoin.util.crypto.**cache_key**(***kw*)
Calculate a cache key (ascii only)

Important key properties:

•The key must be different for different <kw>.

•Key is pure ascii

Parameters **kw** – keys/values to compute cache key from

MoinMoin.util.crypto.**generate_token**(*key=None*, *stamp=None*)
generate a pair of a secret key and a crypto token.

you can use this to implement a password recovery functionality by calling generate_token() and transmitting the returned token to the (correct) user (e.g. by email) and storing the returned (secret) key into the user's profile on the server side.

after the user received the token, he returns to the wiki, gives his user name or email address and the token he received. read the (secret) key from the user profile and call valid_token(key, token) to verify if the token is valid. if it is, consider the user authenticated, remove the secret key from his profile and let him reset his password.

Parameters

- **key** – give it to recompute some specific token for verification
- **stamp** – give it to recompute some specific token for verification

Return type 2-tuple

Returns key, token (both unicode)

MoinMoin.util.crypto.**make_uuid**()

MoinMoin.util.crypto.**random_string**(*length*, *allowed_chars=None*)
Generate a random string with given length consisting of the given characters.

Note: this is now just a little wrapper around passlib's randomness code.

Parameters

- **length** – length of the string
- **allowed_chars** – string with allowed characters or None to indicate all 256 byte values should be used

Returns random string

MoinMoin.util.crypto.**valid_token**(*key*, *token*, *timeout=7200*)
check if token is valid with respect to the secret key, the token must not be older than timeout seconds.

Parameters

- **key** – give the secret key to verify the token
- **token** – the token to verify
- **timeout** – timeout seconds, set to None to ignore timeout

Return type bool

Returns token is valid and not timed out

MoinMoin.util.diff3 module

MoinMoin - diff3 algorithm

MoinMoin.util.diff3.**find_match**(*list1*, *list2*, *nr1*, *nr2*, *mincount=3*)
searches next matching pattern with lenght mincount if no pattern is found len of the both lists is returned

MoinMoin.util.diff3.**main**()

MoinMoin.util.diff3.**match**(*list1*, *list2*, *nr1*, *nr2*, *maxcount=3*)
return the number matching items after the given positions maximum maxcount lines are are processed

MoinMoin.util.diff3.**merge**(*old*, *other*, *new*, *allow_conflicts=1*, **markers*)
do line by line diff3 merge input must be lists containing single lines

MoinMoin.util.diff3.**text_merge**(*old*, *other*, *new*, *allow_conflicts=1*, **markers*)
do line by line diff3 merge with three strings

MoinMoin.util.diff3.**tripple_match**(*old*, *other*, *new*, *other_match*, *new_match*)
find next matching pattern unchanged in both other and new return the position in all three lists

MoinMoin.util.diff_datastruct module

class MoinMoin.util.diff_datastruct.**UndefinedType**
Bases: object

Represents a non-existing value

MoinMoin.util.diff_datastruct.**diff**(*d1*, *d2*, *basekeys=None*)
Get the diff of 2 datastructures (usually 2 meta dicts)

Parameters

- **d1** – old datastructure
- **d2** – new datastructure
- **basekeys** – list of data keys' basenames (default: None, meaning [])

Returns a list of tuples of the format (<change type>, <basekeys>, <value>) that can be used to format a diff

MoinMoin.util.diff_datastruct.**make_text_diff**(*changes*)
Transform change tuples into text diffs

Parameters **changes** – a list of tuples of the format (<change type>, <basekeys>, <value>) that represent a diff

Returns a generator of text diffs

MoinMoin.util.diff_html module

MoinMoin - Side by side diffs

MoinMoin.util.diff_html.**diff**(*old*, *new*)
Find changes between old and new and return HTML markup visualising them.

Parameters

- **old** – old text [unicode]
- **new** – new text [unicode]

MoinMoin.util.diff_html.**indent**(*line*)

MoinMoin.util.diff_text module

MoinMoin - simple text diff (uses difflib)

MoinMoin.util.diff_text.**diff**(*oldlines*, *newlines*, ***kw*)
Find changes between oldlines and newlines.

Parameters

- **oldlines** – list of old text lines
- **newlines** – list of new text lines
- **ignorews** – if 1: ignore whitespace

Return type *list*

Returns lines like diff tool does output.

MoinMoin.util.filesys module

MoinMoin - File System Utilities

MoinMoin.util.filesys.**access_denied_decorator**(*fn*)
Due to unknown reasons, some os.* functions on Win32 sometimes fail with Access Denied (although access should be possible). Just retrying it a bit later works and this is what we do.

MoinMoin.util.filesys.**chmod**(*name*, *mode*, *catchexception=True*)
change mode of some file/dir on platforms that support it.

MoinMoin.util.filesys.**copystat**(*src*, *dst*)
Copy stat bits from src to dst

This should be used when shutil.copystat would be used on directories on win32 because win32 does not support utime() for directories.

According to the official docs written by Microsoft, it returns ENOACCES if the supplied filename is a directory. Looks like a trainee implemented the function.

MoinMoin.util.filesys.**copytree**(*src*, *dst*, *symlinks=False*)
Recursively copy a directory tree using copy2().

The destination directory must not already exist. If exception(s) occur, an Error is raised with a list of reasons.

If the optional symlinks flag is true, symbolic links in the source tree result in symbolic links in the destination tree; if it is false, the contents of the files pointed to by symbolic links are copied.

In contrary to shutil.copytree, this version also copies directory stats, not only file stats.

MoinMoin.util.filesys.**fuid**(*filename*, *max_staleness=3600*)
return a unique id for a file

Using just the file's mtime to determine if the file has changed is not reliable - if file updates happen faster than the file system's mtime granularity, then the modification is not detectable because the mtime is still the same.

This function tries to improve by using not only the mtime, but also other metadata values like file size and inode to improve reliability.

For the calculation of this value, we of course only want to use data that we can get rather fast, thus we use file metadata, not file data (file content).

Note: depending on the operating system capabilities and the way the file update is done, this function might return the same value even if the file has changed. It should be better than just using file's mtime though. max_staleness tries to avoid the worst for these cases.

Parameters

- **filename** – file name of the file to look at
- **max_staleness** – if a file is older than that, we may consider it stale and return a different uid - this is a dirty trick to work around changes never being detected. Default is 3600 seconds, use None to disable this trickery. See below for more details.

Returns an object that changes value if the file changed, None is returned if there were problems accessing the file

MoinMoin.util.filesys.**rename_no_overwrite**(*oldname*, *newname*, *delete_old=False*)
Multiplatform rename

This kind of rename is doing things differently: it fails if newname already exists. This is the usual thing on win32, but not on posix.

If delete_old is True, oldname is removed in any case (even if the rename did not succeed).

MoinMoin.util.filesys.**touch**(*name*)

MoinMoin.util.forms module

MoinMoin - form helpers for flatland / jinja2

class MoinMoin.util.forms.**FileStorage**(*value=Unspecified*, ***kw*)
Bases: flatland.schema.scalars.Scalar

Schema element for Werkzeug FileStorage instances.

adapt(*value*)

MoinMoin.util.forms.**autofocus_filter**(*tagname*, *attributes*, *contents*, *context*, *bind*)

MoinMoin.util.forms.**button_filter**(*tagname*, *attributes*, *contents*, *context*, *bind*)
Show translated text in clickable buttons and submits.

MoinMoin.util.forms.**error_filter**(*tagname*, *attributes*, *contents*, *context*, *bind*)

MoinMoin.util.forms.**error_filter_factory**(*class_='moin-error'*)
Returns an HTML generation filter annotating field CSS class on error.

Parameters **class** – The css class to apply in case of validation error on a field. Default: 'error'

MoinMoin.util.forms.**label_filter**(*tagname*, *attributes*, *contents*, *context*, *bind*)
Provide a translated, generated fallback for field labels.

MoinMoin.util.forms.**make_generator**()
make an html generator

MoinMoin.util.forms.**placeholder_filter**(*tagname*, *attributes*, *contents*, *context*, *bind*)

MoinMoin.util.forms.**required_filter**(*tagname*, *attributes*, *contents*, *context*, *bind*)

MoinMoin.util.interwiki module

MoinMoin - interwiki support code

class MoinMoin.util.interwiki.**CompositeName**
Bases: *MoinMoin.util.interwiki.CompositeName*

namedtuple to hold the compositename

fullname

get_root_fqname()
Set value to the item_root of that namespace, and return the new CompisteName.

query
returns a dict that can be used as a whoosh query to lookup index documents matching this CompositeName

split
returns a dict of field_names/field_values

class MoinMoin.util.interwiki.**InterWikiMap**(*s*)
Bases: object

Parse a valid interwiki map file/string, transforming into a simple python dict object. Provides a set of utilities for parsing and checking a interwiki maps.

SKIP = '#'

static from_file(*filename*)
Load and parse a valid interwiki map file.

static from_string(*ustring*)
Load and parse a valid interwiki map "unicode" object.

MoinMoin.util.interwiki.**getInterwikiHome**(*username*)
Get a user's homepage.

cfg.user_homewiki influences behaviour of this: 'Self' does mean we store user homepage in THIS wiki. When set to our own interwikiname, it behaves like with 'Self'.

'SomeOtherWiki' means we store user homepages in another wiki.

Parameters **username** – the user's name

Return type tuple

Returns (wikiname, itemname)

MoinMoin.util.interwiki.**getInterwikiName**(*item_name*)
Get the (fully qualified) interwiki name of a local item name.

Parameters **item_name** – item name (unicode)

Return type unicode

Returns wiki_name:item_name

MoinMoin.util.interwiki.**get_download_file_name**(*fqname*)
returns the filename that is used for downloading items

MoinMoin.util.interwiki.**get_fqname**(*item_name*, *field*, *namespace*)
Compute composite name from item_name, field, namespace composite name == [NAMESPACE/][@FIELD/]NAME

MoinMoin.util.interwiki.**is_known_wiki**(*wiki_name*)
check if <wiki_name> is a known wiki name

Note: interwiki_map should have entries for the special wikinames denoting THIS wiki, so we do not need to check these names separately.

MoinMoin.util.interwiki.**is_local_wiki**(*wiki_name*)
check if <wiki_name> is THIS wiki

MoinMoin.util.interwiki.**join_wiki**(*wikiurl*, *wikitail*, *field*, *namespace*)

Add a (url_quoted) page name to an interwiki url.

Note: We can't know what kind of URL quoting a remote wiki expects. We just use a utf-8 encoded string with standard URL quoting.

Parameters

- **wikiurl** – wiki url, maybe including a $PAGE placeholder
- **wikitail** – page name
- **namespace** – namespace

Return type string

Returns generated URL of the page in the other wiki

MoinMoin.util.interwiki.**split_fqname**(*url*)

Split a fully qualified url into namespace, field and pagename url -> [NAMESPACE/][@FIELD/]NAME

Parameters **url** – the url to split

Returns a namedtuple CompositeName(namespace, field, itemname)

Examples:

```
url: u'ns1/ns2/@itemid/Page' return u'ns1/ns2', u'itemid', u'Page'
url: u'@revid/OtherPage' return u'', u'revid', u'OtherPage'
url: u'ns1/Page' return u'ns1', u'', u'Page'
url: u'ns1/ns2/@notfield' return u'ns1/ns2', u'', u'@notfield'
```

MoinMoin.util.interwiki.**split_interwiki**(*wikiurl*)

Split a interwiki name, into wikiname and pagename, e.g:

```
'MoinMoin/FrontPage' -> "MoinMoin", "", "", "FrontPage"
'FrontPage' -> "Self", "", "", "FrontPage"
'MoinMoin/Page with blanks' -> "MoinMoin", "", "", "Page with blanks"
'MoinMoin/' -> "MoinMoin", "", "", ""
'MoinMoin/@Someid/SomeValue' -> "MoinMoin", "", "Someid", "SomeValue" if␣
↪Someid field exists or "MoinMoin", "", "", "Someid/SomePage" if not
'MoinMoin/interwikins/AnyPage' -> "MoinMoin", "interwikins", "", "AnyPage"
'ns/AnyPage' -> "Self", "ns", "", "AnyPage" if ns namespace exists or "Self", "
↪", "", "ns:AnyPage" if not.
'ns1/ns2/AnyPage' -> "Self", "ns1/ns2", "", "AnyPage" if ns1/ns2 namespace␣
↪exists OR
                     "Self", "ns1", "", "ns2/AnyPage" if ns1 namespace exists␣
↪OR
                     "Self", "", "", "ns1/ns2/AnyPage" else.
'MoinMoin/ns/@Somefield/AnyPage' -> "MoinMoin", "ns", "", "@Somefield/AnyPage"␣
↪if ns namespace exists and field Somefield does not OR
                                    "MoinMoin", "ns", "Somefield", "AnyPage" if␣
↪ns namespace and field Somefield exist OR
                                    "MoinMoin", "", "", "ns/@Somefield/AnyPage"␣
↪else.
:param wikiurl: the url to split
:rtype: tuple
:returns: (wikiname, namespace, field, pagename)
```

MoinMoin.util.interwiki.**url_for_item**(*item_name*, *wiki_name=u''*, *field=u''*, *namespace=u''*, *rev=u'current'*, *endpoint=u'frontend.show_item'*, *_external=False*)

Compute URL for some local or remote/interwiki item.

For local items: give <rev> to get the url of some specific revision. give the <endpoint> to get the url of some specific view, give _external=True to compute fully specified URLs.

For remote/interwiki items: If you just give <item_name> and <wiki_name>, a generic interwiki URL will be built. If you also give <rev> and/or <endpoint>, it is assumed that remote wiki URLs are built in the same way as local URLs. Computed URLs are always fully specified.

MoinMoin.util.iri module

MoinMoin - Generic? IRI implementation

Implements the generic IRI form as defined in RFC 3987.

class MoinMoin.util.iri.**Iri**(*_iri=None*, *_quoted=True*, *scheme=None*, *authority=None*, *path=None*, *query=None*, *fragment=None*)

Bases: *MoinMoin.util.pysupport.AutoNe*

authority
Authority part of the IRI.

fragment
Fragment part of the IRI.

overall_rules = '\n ^\n (\n (?P<scheme>\n [^:/?\\#]+\n)\n :\n)?\n (\n //\n (?P<authority>\n [^/?\\#]*\n)\n)?\n (?P

path
Path part of the IRI.

query
Query part of the IRI.

scheme
Scheme part of the IRI.

class MoinMoin.util.iri.**IriAuthority**(*iri_authority=None*, *_quoted=True*, *userinfo=None*, *host=None*, *port=None*)

Bases: *MoinMoin.util.pysupport.AutoNe*

authority_rules = '\n ^\n (\n (?P<userinfo>\n [^@]*\n)\n @\n)?\n (?P<host>\n .*?\n)\n (\n :\n (?P<port>\n \\d*

fullquoted
Full quoted form of the authority part of the IRI.

All characters which are illegal in the authority part are encoded. Used to generate the full IRI.

host

quoted
Minimal quoted form of the authority part of the IRI.

Only '%' and illegal UTF-8 sequences are encoded. Primarily used to have a one-to-one mapping with non-UTF-8 URIs.

urlquoted
URI quoted form of the authority part of the IRI.

All characters which are illegal in the authority part are encoded. Used to generate the full URI.

userinfo

class MoinMoin.util.iri.**IriAuthorityHost**
Bases: MoinMoin.util.iri._Value

class MoinMoin.util.iri.**IriAuthorityUserinfo**
Bases: MoinMoin.util.iri._Value

class MoinMoin.util.iri.**IriFragment**
Bases: MoinMoin.util.iri._Value

quote_filter = frozenset(['@', ':', '?', '/'])

class MoinMoin.util.iri.**IriPath**(*iri_path=None*, *_quoted=True*)
Bases: *MoinMoin.util.pysupport.AutoNe*

extend(*value*)

fullquoted
Full quoted form of the path part of the IRI.

All characters which are illegal in the path part are encoded. Used to generate the full IRI.

quoted
Minimal quoted form of the path part of the IRI.

Only '%' and illegal UTF-8 sequences are encoded. Primarily used to have a one-to-one mapping with non-UTF-8 URIs.

urlquoted
URI quoted form of the path part of the IRI.

All characters which are illegal in the path part are encoded. Used to generate the full URI.

class MoinMoin.util.iri.**IriPathSegment**
Bases: MoinMoin.util.iri._Value

quote_filter = frozenset(['@', ':', '/'])

class MoinMoin.util.iri.**IriQuery**
Bases: MoinMoin.util.iri._Value

quote_filter = frozenset(['@', ':', '?', '/'])

MoinMoin.util.mime module

MoinMoin - MIME helpers

class MoinMoin.util.mime.**Type**
Bases: *MoinMoin.util.mime.Type*, *MoinMoin.util.pysupport.AutoNe*

Variables

- **type** – Type part
- **subtype** – Subtype part
- **parameters** – Parameters part

issupertype(*other*)
Check if this object is a super type of the other

A super type is defined as - the other type matches this (possibly wildcard) type, - the other subtype matches this (possibly wildcard) subtype and - the other parameters are a supperset of this one.

s = frozenset(['!', '#', '%', '$', '"', '&', '+', '*', '-', '.', '1', '0', '3', '2', '5', '4', '7', '6', '9', '8', 'A', 'C', 'B', 'E', 'D', 'G

MoinMoin.util.mimetype module

MoinMoin - mimetype support

class MoinMoin.util.mimetype.**MimeType**(*mimestr=None*, *filename=None*)
Bases: object

represents a mimetype like text/plain

as_attachment(*cfg*)

content_type(*major=None*, *minor=None*, *charset=None*, *params=None*)
return a string suitable for Content-Type header

mime_type()
return a string major/minor only, no params

module_name()
convert this mimetype to a string useable as python module name, we yield the exact module name first and then proceed to shorter module names (useful for falling back to them, if the more special module is not found) - e.g. first "text_python", next "text". Finally, we yield "application_octet_stream" as the most general mimetype we have.

Hint: the fallback handler module for text/* should be implemented in module "text" (not "text_plain")

parse_filename(*filename*)

parse_format(*format*)
maps from what we currently use on-page in a #format xxx processing instruction to a sanitized mimetype major, minor tuple. can also be user later for easier entry by the user, so he can just type "wiki" instead of "text/x.moin.wiki".

parse_mimetype(*mimestr*)
take a string like used in content-type and parse it into components, alternatively it also can process some abbreviated string like "wiki"

sanitize()
convert to some representation that makes sense - this is not necessarily conformant to /etc/mime.types or IANA listing, but if something is readable text, we will return some `text/*` mimetype, not `application/*`, because we need text/plain as fallback and not application/octet-stream.

spoil()
this returns something conformant to /etc/mime.type or IANA as a string, kind of inverse operation of sanitize(), but doesn't change self

MoinMoin.util.monkeypatch module

This module contains some monkeypatching for 3rd party code we use.

We hope that any 3rd party might find this code useful and will adopt it, so we don't need to patch it any more. If you adopt some code from here, please notify us, so we can remove it from here.

class MoinMoin.util.monkeypatch.**BaseRequestHandler**(*request*, *client_address*, *server*)
Bases: werkzeug.serving.WSGIRequestHandler

log(*type*, *message*, **args*)

MoinMoin.util.notifications module

MoinMoin - Notifications

class MoinMoin.util.notifications.**Notification**(*app*, *fqname*, *revs*, ***kwargs*)
Bases: object

Represents a mail notification about an item change

generate_diff_url(*domain*)
Generate the URL that leads to diff page of the last 2 revisions

Parameters **domain** – domain name

Returns the absolute URL to the diff page

get_content_diff()
Create a content diff for the last item change

Returns list of diff lines

get_meta_diff()
Create a meta diff for the last item change

Returns a list of tuples of the format (<change type>, <basekeys>, <value>) that can be used to format a diff

html_template = 'mail/notification_main.html'

render_templates(*content_diff*, *meta_diff*)
Render both plain text and HTML templates by providing all the necessary arguments

Returns tuple consisting of plain text and HTML notification message

txt_template = 'mail/notification.txt'

MoinMoin.util.notifications.**get_item_last_revisions**(*app*, *fqname*)
Get 2 or less most recent item revisions from the index

Parameters

- **app** – local proxy app
- **fqname** – the fqname of the item

Returns a list of revisions

MoinMoin.util.notifications.**msgs**()
Encapsulates the main notification messages

Returns a dictionary of notification messages

MoinMoin.util.notifications.**send_notifications**(*app*, *fqname*, ***kwargs*)
Send mail notifications to subscribers on item change

Parameters

- **app** – local proxy app
- **fqname** – fqname of the changed item
- **kwargs** – key/value pairs that contain extra information about the item required in order to create a notification

MoinMoin.util.paramparser module

MoinMoin - parameter parsing and invoking of extension functions

exception MoinMoin.util.paramparser.**BracketError**
Bases: exceptions.Exception

exception MoinMoin.util.paramparser.**BracketMissingCloseError**(*bracket*)
Bases: *MoinMoin.util.paramparser.BracketError*

exception MoinMoin.util.paramparser.**BracketUnexpectedCloseError**(*bracket*)
Bases: *MoinMoin.util.paramparser.BracketError*

class MoinMoin.util.paramparser.**IEFArgument**
Base class for new argument parsers for invoke_extension_function.

get_default()
Return the default for this argument.

parse_argument(*s*)
Parse the argument given in s (a string) and return the argument for the extension function.

class MoinMoin.util.paramparser.**ParserPrefix**(*prefix*)
Bases: object

Trivial container-class holding a single character for the possible prefixes for parse_quoted_separated_ext and implementing rich equal comparison.

class MoinMoin.util.paramparser.**UnitArgument**(*default, argtype, units=['mm'], defaultunit=None*)

Bases: *MoinMoin.util.paramparser.IEFArgument*

Argument class for invoke_extension_function that forces having any of the specified units given for a value.

Note that the default unit is "mm".

Use, for example, "UnitArgument('7mm', float, ['%', 'mm'])".

If the defaultunit parameter is given, any argument that can be converted into the given argtype is assumed to have the default unit. NOTE: This doesn't work with a choice (tuple or list) argtype.

get_default()

parse_argument(*s*)

MoinMoin.util.paramparser.**get_bool**(*arg*, *name=None*, *default=None*)

For use with values returned from parse_quoted_separated or given as macro parameters, return a boolean from a unicode string. Valid input is 'true'/'false', 'yes'/'no' and '1'/'0' or None for the default value.

Parameters

- **arg** – The argument, may be None or a unicode string
- **name** – Name of the argument, for error messages
- **default** – default value if arg is None

Return type boolean or None

Returns the boolean value of the string according to above rules (or default value)

MoinMoin.util.paramparser.**get_choice**(*arg, name=None, choices=[None], default_none=False*)

For use with values returned from parse_quoted_separated or given as macro parameters, return a unicode string that must be in the choices given. None is a valid input and yields first of the valid choices.

Parameters

- **arg** – The argument, may be None or a unicode string
- **name** – Name of the argument, for error messages
- **choices** – the possible choices
- **default_none** – If False (default), get_choice returns first available choice if arg is None. If True, get_choice returns None if arg is None. This is useful if some arg value is required (no default choice).

Return type unicode or None

Returns the unicode string (or default value)

MoinMoin.util.paramparser.**get_complex**(*arg*, *name=None*, *default=None*)

For use with values returned from parse_quoted_separated or given as macro parameters, return a complex from a unicode string. None is a valid input and yields the default value.

Parameters

- **arg** – The argument, may be None or a unicode string
- **name** – Name of the argument, for error messages
- **default** – default return value if arg is None

Return type complex or None

Returns the complex value of the string (or default value)

MoinMoin.util.paramparser.**get_float**(*arg*, *name=None*, *default=None*)

For use with values returned from parse_quoted_separated or given as macro parameters, return a float from a unicode string. None is a valid input and yields the default value.

Parameters

- **arg** – The argument, may be None or a unicode string
- **name** – Name of the argument, for error messages
- **default** – default return value if arg is None

Return type float or None

Returns the float value of the string (or default value)

MoinMoin.util.paramparser.**get_int**(*arg*, *name=None*, *default=None*)

For use with values returned from parse_quoted_separated or given as macro parameters, return an integer from a unicode string containing the decimal representation of a number. None is a valid input and yields the default value.

Parameters

- **arg** – The argument, may be None or a unicode string
- **name** – Name of the argument, for error messages
- **default** – default value if arg is None

Return type int or None

Returns the integer value of the string (or default value)

MoinMoin.util.paramparser.**get_unicode**(*arg*, *name=None*, *default=None*)

For use with values returned from parse_quoted_separated or given as macro parameters, return a unicode string from a unicode string. None is a valid input and yields the default value.

Parameters

- **arg** – The argument, may be None or a unicode string
- **name** – Name of the argument, for error messages
- **default** – default return value if arg is None;

Return type unicode or None

Returns the unicode string (or default value)

MoinMoin.util.paramparser.**invoke_extension_function**(*function*, *args*, *fixed_args=[]*)

Parses arguments for an extension call and calls the extension function with the arguments.

If the macro function has a default value that is a bool, int, long, float or unicode object, then the given value is converted to the type of that default value before passing it to the macro function. That way, macros need not call the get_* functions for any arguments that have a default.

Parameters

- **function** – the function to invoke
- **args** – unicode string with arguments (or evaluating to False)
- **fixed_args** – fixed arguments to pass as the first arguments

Returns the return value from the function called

MoinMoin.util.paramparser.**parse_quoted_separated**(*args*, *separator=','*, *name_value=True*, *seplimit=0*)

MoinMoin.util.paramparser.**parse_quoted_separated_ext**(*args*, *separator=None*, *name_value_separator=None*, *brackets=None*, *seplimit=0*, *multikey=False*, *prefixes=None*, *quotes=""*)

Parses the given string according to the other parameters.

Items can be quoted with any character from the quotes parameter and each quote can be escaped by doubling it, the separator and name_value_separator can both be quoted, when name_value_separator is set then the name can also be quoted.

Values that are not given are returned as None, while the empty string as a value can be achieved by quoting it.

If a name or value does not start with a quote, then the quote looses its special meaning for that name or value, unless it starts with one of the given prefixes (the parameter is unicode containing all allowed prefixes.) The prefixes will be returned as ParserPrefix() instances in the first element of the tuple for that particular argument.

If multiple separators follow each other, this is treated as having None arguments inbetween, that is also true for when space is used as separators (when separator is None), filter them out afterwards.

The function can also do bracketing, i.e. parse expressions that contain things like:

```
"(a (a b))" to ['(', 'a', ['(', 'a', 'b']],
```

in this case, as in this example, the returned list will contain sub-lists and the brackets parameter must be a list of opening and closing brackets, e.g.:

```
brackets = ['()', '<>']
```

Each sub-list's first item is the opening bracket used for grouping. Nesting will be observed between the different types of brackets given. If bracketing doesn't match, a BracketError instance is raised with a 'bracket' property indicating the type of missing or unexpected bracket, the instance will be either of the class BracketMissingCloseError or of the class BracketUnexpectedCloseError.

If multikey is True (along with setting name_value_separator), then the returned tuples for (key, value) pairs can also have multiple keys, e.g.:

```
"a=b=c" -> ('a', 'b', 'c')
```

Parameters

- **args** – arguments to parse
- **separator** – the argument separator, defaults to None, meaning any space separates arguments
- **name_value_separator** – separator for name=value, default '=', name=value keywords not parsed if evaluates to False
- **brackets** – a list of two-character strings giving opening and closing brackets
- **seplimit** – limits the number of parsed arguments
- **multikey** – multiple keys allowed for a single value

Return type *list*

Returns list of unicode strings and tuples containing unicode strings, or lists containing the same for bracketing support

class MoinMoin.util.paramparser.**required_arg**(*argtype*)

Wrap a type in this class and give it as default argument for a function passed to invoke_extension_function() in order to get generic checking that the argument is given.

MoinMoin.util.plugins module

MoinMoin - plugin loader

exception MoinMoin.util.plugins.**PluginAttributeError**
Bases: *MoinMoin.util.plugins.PluginError*

Raised when plugin does not contain an attribtue

exception MoinMoin.util.plugins.**PluginError**
Bases: exceptions.Exception

Base class for plugin errors

exception MoinMoin.util.plugins.**PluginMissingError**
Bases: *MoinMoin.util.plugins.PluginError*

Raised when a plugin is not found

MoinMoin.util.plugins.**builtinPlugins**(*kind*)
Gets a list of modules in MoinMoin.'kind'

Parameters **kind** – what kind of modules we look for

Return type *list*

Returns module names

MoinMoin.util.plugins.**getPlugins**(*kind*, *cfg*)
Gets a list of plugin names of kind

Parameters **kind** – what kind of modules we look for

Return type *list*

Returns module names

MoinMoin.util.plugins.**importBuiltinPlugin**(*kind*, *name*, *function='execute'*)
Import builtin plugin from MoinMoin package

See importPlugin docstring.

MoinMoin.util.plugins.**importNameFromPlugin**(*moduleName*, *name*)
Return <name> attr from <moduleName> module, raise PluginAttributeError if name does not exist.

If name is None, return the <moduleName> module object.

MoinMoin.util.plugins.**importPlugin**(*cfg*, *kind*, *name*, *function='execute'*)
Import wiki or builtin plugin

Returns <function> attr from a plugin module <name>. If <function> attr is missing, raise PluginAttributeError. If <function> is None, return the whole module object.

If <name> plugin can not be imported, raise PluginMissingError.

kind may be one of 'action', 'macro' or any other directory that exist in MoinMoin or data/plugin.

Wiki plugins will always override builtin plugins. If you want specific plugin, use either importWikiPlugin or importBuiltinPlugin directly.

Parameters

- **cfg** – wiki config instance
- **kind** – what kind of module we want to import
- **name** – the name of the module
- **function** – the function name

Return type any object

Returns "function" of module "name" of kind "kind", or None

MoinMoin.util.plugins.**importWikiPlugin**(*cfg*, *kind*, *name*, *function='execute'*)
Import plugin from the wiki data directory

See importPlugin docstring.

MoinMoin.util.plugins.**searchAndImportPlugin**(*cfg*, *type*, *name*, *what=None*)

MoinMoin.util.plugins.**wikiPlugins**(*kind*, *cfg*)

Gets a dict containing the names of all plugins of <kind> as the key and the containing module name as the value.

Parameters **kind** – what kind of modules we look for

Return type dict

Returns plugin name to containing module name mapping

MoinMoin.util.profile module

profile - moin profiling utilities

This module provides profilers used to profile the memory usage of a long running process.

Typical usage:

1. Create a profiler:

   from MoinMoin.util.profile import Profiler profiler = Profiler('my log')

2. In the request handler, add each request to the profiler:

   profiler.addRequest()

3. If you like, you can add extra samples:

   profiler.sample()

You can customize the profiler when you create it:

- requestsPerSample (default 100):

  How many requests to run between samples. Set higher for live wiki or lower for more accurate results.

- collect (default 0):

  Use gc.collect to force a memory cleanup each sample. Keeps the memory usage much lower, but your profile data will not reflect the real world memory usage of the application.

Based on code by Oliver Graf

class MoinMoin.util.profile.**Profiler**(*name*, *requestsPerSample=100*, *collect=0*)

Profile memory usage

Profiler count requests and sample memory usage.

FIXME: We might want to save the profiler log in the profiled wiki data dir, but the data dir is available only later in request. This should be fixed by loading the config earlier.

addRequest()

Add a request to the profile

Call this for each page request.

WARNING: This is the most important call. if you don't call this for each request - you will not have any profile data.

Invoke sample when self.count reach self.requestsPerSample.

sample()

Make a sample of memory usage and log it

You can call this to make samples between the samples done each requestsPerSample, for example, at startup.

Invoke common methods for all profilers. Some profilers like TwistedProfiler override this method.

class MoinMoin.util.profile.**TwistedProfiler**(*name*, *requestsPerSample=100*, *collect=0*)
Bases: *MoinMoin.util.profile.Profiler*

Twisted specific memory profiler

Customize the way we call ps, to overcome blocking problems on twisted.

sample()
Make a sample of memory usage and log it

On twisted we can't just call ps - we have to use deferred, which will call us using a callback when its finished, and then we log.

Since twisted continue to serve while the deferred fetch the memory, the reading may be late in few requests.

MoinMoin.util.pysupport module

MoinMoin - Supporting functions for Python magic

class MoinMoin.util.pysupport.**AutoNe**
Bases: object

Implement __ne__ in terms of __eq__. This is a mixin class.

MoinMoin.util.pysupport.**getPackageModules**(*packagefile*)
Return a list of modules for a package, omitting any modules starting with an underscore.

MoinMoin.util.pysupport.**getPluginModules**(*packagedir*)
Return a list of plugin modules for a given plugin package dir, omitting any that start with an underscore.

MoinMoin.util.pysupport.**importName**(*modulename*, *name*)
Import name dynamically from module

Used to do dynamic import of modules and names that you know their names only in runtime.

Any error raised here must be handled by the caller.

Parameters

- **modulename** – full qualified mudule name, e.g. x.y.z
- **name** – name to import from modulename

Return type any object

Returns name from module

MoinMoin.util.pysupport.**isImportable**(*module*)
Check whether a certain module is available.

MoinMoin.util.pysupport.**load_package_modules**(*package_name*, *package_pathes*)
Load (import) all modules from some package (except those starting with _).

This is useful if there is some code in the module that runs at import time and registers some code of that module somewhere.

Call this from __init__.py of the same package like this:

load_package_modules(__name__, __path__)

MoinMoin.util.pysupport.**makeThreadSafe**(*function*, *lock=None*)
Call with a function you want to make thread safe

Call without lock to make the function thread safe using one lock per function. Call with existing lock object if you want to make several functions use same lock, e.g. all functions that change same data structure.

Parameters

- **function** – function to make thread safe

- **lock** – threading.Lock instance or None

Return type function

Returns function decorated with locking

MoinMoin.util.registry module

MoinMoin - Module registry

Every module registers a factory for itself at the registry with a given priority. During the lookup each factory is called with the given arguments and can return a callable to consider itself as a match.

class MoinMoin.util.registry.**Registry**
Bases: *MoinMoin.util.registry.RegistryBase*

register(*factory*, *priority=0*)
Register a factory

Parameters **factory** – Factory to register. Callable, have to return a class

class MoinMoin.util.registry.**RegistryBase**
Bases: object

class Entry
Bases: MoinMoin.util.registry.Entry

RegistryBase.**PRIORITY_FIRST = -10**

RegistryBase.**PRIORITY_LAST = 10**

RegistryBase.**PRIORITY_MIDDLE = 0**

RegistryBase.**PRIORITY_REALLY_FIRST = -20**

RegistryBase.**PRIORITY_REALLY_LAST = 20**

RegistryBase.**get**(**args*, ***kw*)
Lookup a matching module

Each registered factory is called with the given arguments and the first matching wins.

RegistryBase.**unregister**(*factory*)
Unregister a factory

Parameters **factory** – Factory to unregister

MoinMoin.util.rev_navigation module

Helper function to get prior, current, next revisions and mod-times.

MoinMoin.util.rev_navigation.**prior_next_revs**(*revid*, *fqname*)
Return prior, current, and next revids and time stamps.

MoinMoin.util.send_file module

A better send_file

Initially, this was a modified implementation of flask 0.6.0's send_file(), trying to be as compatible as possible.

For details see: https://github.com/mitsuhiko/flask/issues/issue/104 and the history of this file in our repository. This code fixes all the issues described in the bug report.

As we forked send_file, we later modified it (without trying to stay compatible), because we can easily adapt anyway and the code can be much simpler without compatibility code.

MoinMoin.util.send_file.**encode_rfc2231**(*value*, *coding='UTF-8'*, *lang=''*)
Encode a value according to RFC2231/5987.

Parameters

- **value** – the value to encode. must be either unicode or encoded in <coding>.
- **coding** – the coding (charset) to use. it is a good idea to use 'UTF-8'.
- **lang** – the language to use. defaults to empty string (no language given).

MoinMoin.util.send_file.**send_file**(*filename=None*, *file=None*, *mimetype=None*, *as_attachment=False*, *attachment_filename=None*, *mtime=None*, *cache_timeout=43200*, *add_etags=True*, *etag=None*, *conditional=False*)
Sends the contents of a file to the client.

A file can be either a filesystem file or a file-like object (this code is careful about not assuming that every file is a filesystem file).

This will use the most efficient method available, configured and possible (for filesystem files some more optimizations may be possible that for file-like objects not having a filesystem filename). By default it will try to use the WSGI server's file_wrapper support. Alternatively you can set the application's `use_x_sendfile` attribute to `True` to directly emit an *X-Sendfile* header. This however requires support of the underlying webserver for *X-Sendfile*.

send_file will try to guess some stuff for you if you do not provide them:

•mimetype (based on filename / attachment_filename)

•mtime (based on filesystem file's metadata)

•etag (based on filename, mtime, filesystem file size)

If you do not provide enough information, send_file might raise a TypeError.

For extra security you probably want to sent certain files as attachment (HTML for instance).

Please never pass filenames to this function from user sources without checking them first. Something like this is usually sufficient to avoid security problems:

```
if '..' in filename or filename.startswith('/'):
    abort(404)
```

Parameters

- **filename** – the filesystem filename of the file to send (relative to the `root_path` if a relative path is specified). If you just have an open filesystem file object f, give *f.name* here. If you don't have a filesystem file nor a filesystem file name, but just a file-like obj, don't use this argument.
- **file** – a file (or file-like) object, you may give it if you either do not have a filesystem filename or if you already have an open file anyway.
- **mimetype** – the mimetype of the file if provided, otherwise auto detection happens based on the filename or attachment_filename.
- **as_attachment** – set to *True* if you want to send this file with a `Content-Disposition: attachment` header.
- **attachment_filename** – the filename for the attachment if it differs from the filename argument.
- **mtime** – the modification time of the file if provided, otherwise it will be determined automatically for filesystem files
- **cache_timeout** – the timeout in seconds for the headers.
- **conditional** – set to *True* to enable conditional responses.

- **add_etags** – set to *False* to disable attaching of etags.
- **etag** – you can give an etag here, None means to try to compute the etag from the file's filesystem metadata (the latter of course only works for filesystem files). If you do not give a filename, but you use add_etags, you must explicitely provide the etag as it can't compute it for that case.

MoinMoin.util.subscriptions module

MoinMoin - Subscriptions

class MoinMoin.util.subscriptions.**Subscriber**(*itemid*, *name*, *email*, *locale*)
Bases: tuple

email
Alias for field number 2

itemid
Alias for field number 0

locale
Alias for field number 3

name
Alias for field number 1

MoinMoin.util.subscriptions.**get_matched_subscription_patterns**(*subscription_patterns*, ***meta*)
Get all the subscriptions with patterns that match at least one of item names

Parameters

- **subscription_patterns** – a list of subscription patterns (the ones that start with NAMERE or NAMEPREFIX)
- **meta** – key/value pairs from item metadata - name and namespace keys

Returns a list of matched subscription patterns

MoinMoin.util.subscriptions.**get_subscribers**(***meta*)
Get all users that are subscribed to the item

Parameters **meta** – key/value pairs from item metadata - itemid, name, namespace, tags keys

Returns a set of Subscriber objects

MoinMoin.util.thread_monitor module

Thread monitor - Check the state of all threads.

Just call activate_hook() as early as possible in program execution. Then you can trigger the output of tracebacks of all threads by calling trigger_dump().

MoinMoin.util.thread_monitor.**dump_regularly**(*seconds*)
Dumps the tracebacks every 'seconds' seconds.

MoinMoin.util.tree module

MoinMoin - Tree name and element generator

class MoinMoin.util.tree.**Name**
Bases: emeraldtree.tree.QName

Represents a QName and factory for elements with this QName

class MoinMoin.util.tree.**Namespace**
Bases: unicode

Represents a namespace and factory for Names within this namespace

namespace

MoinMoin.util.version module

MoinMoin - dealing with version numbers

class MoinMoin.util.version.**Version**
Bases: tuple

Version objects store versions like 1.2.3a4 in a structured way and support version comparisons and direct version component access. 1: major version (digits only) 2: minor version (digits only) 3: (maintenance) release version (digits only) a4: optional additional version specification (str)

See PEP386 for more details. TODO: use 3rd party code for PEP386 version numbers later.

You can create a Version instance either by giving the components, like: Version(1,2,3,'a4')

or by giving the composite version string, like: Version(version="1.2.3a4").

Version subclasses tuple, so comparisons to tuples should work. Also, we inherit all the comparison logic from tuple base class.

VERSION_RE = <_sre.SRE_Pattern object>

additional

major

minor

classmethod parse_version(*version*)

release

Module contents

MoinMoin - Utility Functions General helper functions that are not directly wiki related.

MoinMoin.util.**TranslateCDATA**(*text*)
Convert a string to a CDATA-encoded one Copyright (c) 1999-2000 FourThought, http://4suite.com/4DOM

MoinMoin.util.**TranslateText**(*text*)
Convert a string to a PCDATA-encoded one (do minimal encoding) Copyright (c) 1999-2000 FourThought, http://4suite.com/4DOM

MoinMoin.util.**getPageContent**(*results*, *offset*, *results_per_page*)
Selects the content to show on a single page

Parameters

- **results** – the whole result, from which results for one page will be selected (generally a generator but could be a list also),
- **offset** – after skipping how many results, the selection of results for that page will be done (int),
- **results_per_page** – number of results to be shown on a single page (int)

Return type tuple

Returns selected_result (list), offset for next page (If 0 then no next page), offset for previous page (If less than 0, then no previous page)

MoinMoin.util.**rangelist**(*numbers*)
Convert a list of integers to a range string in the form '1,2-5,7'.

Submodules

MoinMoin.app module

MoinMoin - wsgi application setup and related code

Use create_app(config) to create the WSGI application (using Flask).

MoinMoin.app.**before_wiki**()
Setup environment for wiki requests, start timers.

MoinMoin.app.**create_app**(*config=None*, *create_index=False*, *create_storage=False*)
simple wrapper around create_app_ext() for flask-script

MoinMoin.app.**create_app_ext**(*flask_config_file=None*, *flask_config_dict=None*, *moin_config_class=None*, *warn_default=True*, ***kwargs*)
Factory for moin wsgi apps

Parameters

- **flask_config_file** – a flask config file name (may have a MOINCFG class), if not given, a config pointed to by MOINCFG env var will be loaded (if possible).
- **flask_config_dict** – a dict used to update flask config (applied after flask_config_file was loaded [if given])
- **moin_config_class** – if you give this, it'll be instantiated as app.cfg, otherwise it'll use MOINCFG from flask config. If that also is not there, it'll use the DefaultConfig built into MoinMoin.
- **warn_default** – emit a warning if moin falls back to its builtin default config (maybe user forgot to specify MOINCFG?)
- **kwargs** – if you give additional keyword args, the keys/values will get patched into the moin configuration class (before its instance is created)

MoinMoin.app.**deinit_backends**(*app*)

MoinMoin.app.**destroy_app**(*app*)

MoinMoin.app.**init_backends**(*app*)
initialize the backends

MoinMoin.app.**setup_user**()
Try to retrieve a valid user object from the request, be it either through the session or through a login.

MoinMoin.app.**teardown_wiki**(*response*)
Teardown environment of wiki requests, stop timers.

MoinMoin.conftest module

MoinMoin Testing Framework

All test modules must be named test_modulename to be included in the test suite. If you are testing a package, name the test module test_package_module.

Tests that require a certain configuration, like section_numbers = 1, must use a Config class to define the required configuration within the test class.

class MoinMoin.conftest.**Module**(*fspath*, *parent=None*, *config=None*, *session=None*)
Bases: _pytest.python.Module

run(**args*, ***kwargs*)

MoinMoin.conftest.**app**(*app_ctx*)

MoinMoin.conftest.**app_ctx**(*cfg*)

MoinMoin.conftest.**cfg**()

MoinMoin.conftest.**pytest_pycollect_makemodule**(*path*, *parent*)

MoinMoin.conftest.**pytest_report_header**(*config*)

MoinMoin.error module

MoinMoin errors / exception classes

exception MoinMoin.error.**CompositeError**(*message*)
Bases: *MoinMoin.error.Error*

Base class for exceptions containing an exception

Do not use this class but its more specific sub classes.

Useful for hiding low level error inside high level user error, while keeping the inner error information for debugging.

Example:

```
class InternalError(CompositeError):
    ''' Raise for internal errors '''

try:
    # code that might fail...
except HairyLowLevelError:
    raise InternalError("Sorry, internal error occurred")
```

When showing a traceback, both InternalError traceback and HairyLowLevelError traceback are available.

exceptions()
Return a list of all inner exceptions

exception MoinMoin.error.**ConfigurationError**(*message*)
Bases: *MoinMoin.error.FatalError*

Raise when fatal misconfiguration is found

exception MoinMoin.error.**Error**(*message*)
Bases: exceptions.Exception

Base class for moin moin errors

Use this class when you raise errors or create sub classes that may be used to display non ASCII error message.

Standard errors work safely only with strings using ascii or unicode. This class can be used safely with both strings using CHARSET and unicode.

You can init this class with either unicode or string using CHARSET encoding. On output, the class will convert the string to unicode or the unicode to string, using CHARSET.

When you want to render an error, use unicode() or str() as needed.

exception MoinMoin.error.**FatalError**(*message*)
Bases: *MoinMoin.error.CompositeError*

Base class for fatal error we can't handle

Do not use this class but its more specific sub classes.

exception MoinMoin.error.**InternalError**(*message*)
Bases: *MoinMoin.error.FatalError*

Raise when internal fatal error is found

MoinMoin.forms module

MoinMoin - Flatland widgets

General Flatland widgets containing hints for the templates.

MoinMoin.forms.**AnyInteger**
alias of Integer

class MoinMoin.forms.**BackReference**(*value=Unspecified*, ***kw*)
Bases: MoinMoin.forms.List

Back references built from Whoosh query.

set(*query*, ***query_args*)

MoinMoin.forms.**Checkbox**
alias of Boolean

MoinMoin.forms.**DateTime**
alias of *DateTimeUNIX*

class MoinMoin.forms.**DateTimeUNIX**(*value=Unspecified*, ***kw*)
Bases: flatland.schema.scalars.DateTime

A DateTime that uses a UNIX timestamp instead of datetime as internal representation of DateTime.

adapt(*value*)
Coerces value to a native UNIX timestamp.

If value is an instance of int and it is a correct UNIX timestamp, returns it unchanged. Otherwise uses DateTime superclass to parse it.

serialize(*value*)
Serializes value to string.

MoinMoin.forms.**Email**
alias of String

class MoinMoin.forms.**Enum**(*value=Unspecified*, ***kw*)
Bases: flatland.schema.scalars.Enum

An Enum with a convenience class method out_of.

classmethod out_of(*choice_specs*, *sort_by=None*)
A convenience class method to build Enum with extra data attached to each valid value.

Parameters

- **choice_specs** – An iterable of tuples. The elements are collected into the choice_specs property; the tuples' first elements become the valid values of the Enum. e.g. for choice_specs = [(v1, ...), (v2, ...), ...], the valid values are v1, v2, ...
- **sort_by** – If not None, sort choice_specs by the sort_by'th element.

MoinMoin.forms.**File**
alias of FileStorage

MoinMoin.forms.**Hidden**
alias of String

MoinMoin.forms.**InlineCheckbox**
alias of Boolean

MoinMoin.forms.**JSON**
alias of String

MoinMoin.forms.**MultiSelect**
alias of Array

MoinMoin.forms.**MultilineText**
alias of String

class MoinMoin.forms.**MyJoinedString**(*value=Unspecified*, ***kw*)
Bases: flatland.schema.compound.JoinedString

A JoinedString that offers the list of children (not the joined string) as value property.

u

value

exception MoinMoin.forms.**NameNotValidError**
Bases: exceptions.ValueError

The name is not valid.

MoinMoin.forms.**Names**
alias of *MyJoinedString*

MoinMoin.forms.**Natural**
alias of Integer

MoinMoin.forms.**OpenID**
alias of String

MoinMoin.forms.**OptionalMultilineText**
alias of String

MoinMoin.forms.**OptionalText**
alias of String

MoinMoin.forms.**Password**
alias of String

MoinMoin.forms.**Quicklinks**
alias of *MyJoinedString*

MoinMoin.forms.**ReadonlyItemLinkList**
alias of List

MoinMoin.forms.**ReadonlyStringList**
alias of List

class MoinMoin.forms.**Reference**(*value=Unspecified*, ***kw*)
Bases: *MoinMoin.forms.Enum*

A metadata property that points to another item selected out of the Results of a search query.

to(*query*, *query_args={}*)

MoinMoin.forms.**RequiredMultilineText**
alias of String

MoinMoin.forms.**RequiredPassword**
alias of String

MoinMoin.forms.**RequiredText**
alias of String

MoinMoin.forms.**Search**
alias of String

MoinMoin.forms.**Select**
alias of *Enum*

MoinMoin.forms.**SelectSubmit**
alias of *Enum*

MoinMoin.forms.**SmallNatural**
alias of Integer

MoinMoin.forms.**Subscriptions**
alias of *SubscriptionsJoinedString*

class MoinMoin.forms.**SubscriptionsJoinedString**(*value=Unspecified*, ***kw*)
Bases: flatland.schema.compound.JoinedString

A JoinedString that offers the list of children as value property and also appends the name of the item to the end of ITEMID subscriptions.

u

value

MoinMoin.forms.**Tags**
alias of *MyJoinedString*

MoinMoin.forms.**Text**
alias of String

MoinMoin.forms.**URL**
alias of String

class MoinMoin.forms.**ValidJSON**(***kw*)
Bases: flatland.validation.base.Validator

Validator for JSON

invalid_itemid_msg = l'Itemid not a proper UUID'

invalid_json_msg = l'Invalid JSON.'

invalid_namespace_msg = ''

validate(*element*, *state*)

validitemid(*itemid*)

validnamespace(*current_namespace*)

class MoinMoin.forms.**ValidName**(***kw*)
Bases: flatland.validation.base.Validator

Validator for Name

invalid_name_msg = ''

validate(*element*, *state*)

class MoinMoin.forms.**ValidReference**(***kw*)
Bases: flatland.validation.base.Validator

Validator for Reference

invalid_reference_msg = l'Invalid Reference.'

validate(*element*, *state*)

MoinMoin.forms.**YourEmail**
alias of String

MoinMoin.forms.**YourOpenID**
alias of String

MoinMoin.forms.**validate_name**(*meta*, *itemid*)
Check whether the names are valid. Will just return, if they are valid, will raise a NameNotValidError if not.

MoinMoin.log module

MoinMoin - init "logging" system

WARNING

logging must be configured VERY early, before the code in log.getLogger gets executed. Thus, logging is configured either by:

1. an environment variable MOINLOGGINGCONF that contains the path/filename of a logging configuration file - this method overrides all following methods (except if it can't read or use that configuration, then it will use c))
2. by an explicit call to MoinMoin.log.load_config('logging.conf') - you need to do this very early or a) or c) will happen before
3. by using a builtin fallback logging conf

If logging is not yet configured, log.getLogger will do an implicit configuration call - then a) or c) is done.

Usage (for wiki server admins)

Either use something like this in some shell script: MOINLOGGINGCONF=/path/to/logging.conf export MOINLOGGINGCONF

Or, modify your server adaptor script (e.g. moin.cgi) to do this:

```
from MoinMoin import log
log.load_config('wiki/config/logging/logfile') # XXX please fix this path!
```

You have to fix that path to use a logging configuration matching your needs (we provide some examples in the path given there, it is relative to the uncompressed moin distribution archive - if you use some moin package, you maybe find it under /usr/share/moin/). It is likely that you also have to edit the sample logging configurations we provide (e.g. to fix the logfile location).

Usage (for developers)

If you write code for moin, do this at top of your module:

```
from MoinMoin import log
logging = log.getLogger(__name__)
```

This will create a logger with 'MoinMoin.your.module' as name. The logger can optionally get configured in the logging configuration. If you don't configure it, some upperlevel logger (e.g. the root logger) will do the logging.

class MoinMoin.log.**EmailHandler**(*toaddrs=[]*, *subject=u''*)
Bases: logging.Handler

A custom handler class which sends email for each logging event using wiki mail configuration

emit(*record*)
Emit a record.

Send the record to the specified addresses

MoinMoin.log.**getLogger**(*name*)
wrapper around logging.getLogger, so we can do some more stuff:

- preprocess logger name
- patch loglevel constants into logger object, so it can be used instead of the logging module

MoinMoin.log.**load_config**(*conf_fname=None*)
load logging config from conffile

MoinMoin.user module

MoinMoin - User Accounts

TODO: Currently works on unprotected user backend

This module contains functions to access user accounts (list all users, get some specific user). User instances are used to access the user profile of some specific user (name, password, email, bookmark, trail, settings, ...).

class MoinMoin.user.**User**(*uid=None*, *name=''*, *password=None*, *auth_username=''*, *trusted=False*, ***kw*)
Bases: object

A MoinMoin User

add_trail(*item_name*)
Add item name to trail.

Parameters **item_name** – the item name (unicode) to add to the trail

apply_recovery_token(*token*, *newpass*)

avatar(*size=30*)

bookmark
Get bookmark timestamp.

Return type int / None

Returns bookmark timestamp or None

create_or_update(*changed=False*)
Create or update a user profile

Parameters **changed** – bool, set this to True if you updated the user profile values

exists()
Do we have a user profile for this user?

Return type bool

Returns true, if we have a user account

generate_recovery_token()

generate_session_token(*save=True*)
Generate new session token and key pair. Used to validate sessions.

getText(*text*)
translate a text to the language of this user

get_session_token()
Get current session token. If there is no token, generate a new one.

get_trail()
Return list of recently visited item names.

Return type *list*

Returns item names (unicode) in trail

has_invalidated_password()
Check if the password hash of this user is invalid.

is_current_user()
Check if this user object is the user doing the current request

is_quicklinked_to(*pagelist*)
Check if user quicklink matches any page in pagelist.

Parameters **pagelist** – list of pages to check for quicklinks

Return type bool

Returns if user has quicklinked any page in pagelist

is_subscribed_to(*item*)
Check if user is subscribed to the following item

Parameters **item** – Item object

Return type bool

Returns if user is subscribed to the item

language

load_from_id(*itemid*, *password=None*)
Load user account data from disk.

Parameters **password** – If not None, then the given password must match the password in the user account file.

logout_session(*all_browsers=True*)
Terminate session in all browsers unless all_browsers is set to False

mail_email_verification()
Mail a user a link to verify his email address.

mail_password_recovery(*cleartext_passwd=None*, *subject=None*, *text=None*)
Mail a user who forgot his password a message enabling him to login again.

name0

quicklink(*pagename*)
Adds a page to the user quicklinks

Add links as interwiki names.

Parameters **pagename** (*unicode*) – page name

Return type bool

Returns if pagename was added

quickunlink(*pagename*)
Remove a page from user quicklinks

Remove interwiki name from quicklinks.

Parameters **pagename** (*unicode*) – page name

Return type bool

Returns if pagename was removed

save(*force=False*)
Save user account data to user account file on disk.

set_password(*password*, *is_encrypted=False*, *salt=None*)
Set or update the password (hash) stored for this user.

Parameters

- **password** – the new password (or pw hash) giving an empty string or None as password will invalidate the stored password hash (meaning that it will not match against any given password)

- **is_encrypted** – if False (default), the password is given as plaintext and will be "encrypted" (hashed) before getting stored. if True, the already "encrypted" password hash is given in param password and will be stored "as is" - this is mainly useful for tests.
- **salt** – if None (default), passlib will generate and use a random salt. Otherwise, the given salt will be used - this is mainly useful for tests.

subscribe(*keyword*, *value*, *namespace=None*)
Subscribe to a wiki page.

The user can subscribe in 5 different ways:

•by itemid - ITEMID:<itemid value>

•by item name - NAME:<namespace>:<name value>

•by a tagname - TAGS:<namespace>:<tag value>

•by a prefix name - NAMEPREFIX:<namespace>:<name prefix>

•by a regular expression - NAMERE:<namespace>:<name regexp>

Parameters

- **keyword** – the keyword (itemid, name, tags, nameprefix, namere) by which the type of the subscription is determined
- **value** – the subscription value (itemid, name, tag, regexp or nameprefix value)
- **namespace** – the namespace of the subscription; itemid keyword doesn't require a namespace

Return type bool

Returns if user was subscribed

unsubscribe(*keyword*, *value*, *namespace=None*, *item=None*)
Unsubscribe from a wiki page.

Same as for subscribing, user can also unsubscribe in 5 ways. The unsubscribe action doesn't guarantee that user will not receive any notification for this item, since user can be subscribed by some other patterns that match current item.

Parameters

- **keyword** – the keyword (itemid, name, tags, nameprefix, namere) by which the type of the subscription is determined
- **value** – the subscription value (itemid, name, tag, regexp or nameprefix value)
- **namespace** – the namespace of the subscription; itemid keyword doesn't require a namespace
- **item** – Item object to check if the user is still subscribed

Return type bool

Returns if user was unsubscribed

validate_recovery_token(*token*)

validate_session(*token*)
Check if the session token is valid.

Invalid session tokens happen for these cases:

1. there are multiple sessions (different machines, different browsers) open for same user. the user then changes the password in one of these, which creates a new session key in the profile also, which invalidates all sessions everywhere else for this user.

2.the user profile is gone (e.g. due to erasing the storage), then a invalid session key will be read from the profile (from cfg.user_defaults) that will never validate against the session key read from the session.

class MoinMoin.user.**UserProfile**(**q)
Bases: object

A User Profile

load(**q)
load a user profile, the query q can use any indexed (unique) field

save(*force=False*)
save a user profile (if it was changed since loading it)

Note: if mutable profile values were modified, you need to use force=True because these changes are not detected!

stored

MoinMoin.user.**assemble_subscription**(*keyword*, *value*, *namespace=None*)
Create a valid subscription string

Parameters

- **keyword** – the keyword (itemid, name, tags, nameprefix, namere) by which the type of the subscription is determined
- **value** – the subscription value (itemid, name, tag, regexp or nameprefix value)
- **namespace** – the namespace of the subscription

Returns subscription string

MoinMoin.user.**create_user**(*username*, *password*, *email*, *validate=True*, *is_encrypted=False*, *verify_email=False*, **meta)
Create a new user

Parameters

- **username** – unique user name
- **password** – user's password - see also is_encrypted param
- **email** – unique email address
- **validate** – if True (default) will validate username, password, email and the uniqueness of the user created
- **is_encrypted** – if False (default) defines that the password is in plaintext, when True - password was already encrypted
- **meta** – a dictionary of key-value pairs that represent user metadata and will be stored into user profile metadata

Verify_email if True email is saved in user.profile[EMAIL_UNVALIDATED], else email is saved in user.profile[EMAIL]

MoinMoin.user.**get_editor**(*userid*, *addr*, *hostname*)
Return a tuple of type id and string or Page object representing the user that did the edit.

The type id is one of 'ip' (DNS or numeric IP), 'email' (email addr), 'interwiki' (Interwiki homepage) or 'anon' ('').

MoinMoin.user.**get_user_backend**()

MoinMoin.user.**isValidName**(*name*)
Validate user name

Parameters **name** – user name, unicode

MoinMoin.user.**normalizeName**(*name*)
Make normalized user name

Prevent impersonating another user with names containing leading, trailing or multiple whitespace, or using invisible unicode characters.

Prevent creating user page as sub page, because '/' is not allowed in user names.

Prevent using ':' and ',' which are reserved by acl.

Parameters **name** – user name, unicode

Return type unicode

Returns user name that can be used in acl lines

MoinMoin.user.**search_users**(**q)
Searches for a users with given query keys/values

MoinMoin.user.**update_user_query**(**q)

MoinMoin.wikiutil module

MoinMoin - Wiki Utility Functions

MoinMoin.wikiutil.**AbsItemName**(*context*, *itemname*)
Return the absolute item name for a (possibly) relative item name.

Parameters

- **context** – name of the item where "itemname" appears on
- **itemname** – the (possibly relative) item name

Return type unicode

Returns the absolute item name

MoinMoin.wikiutil.**ParentItemName**(*itemname*)
Return the parent item name.

Parameters **itemname** – the absolute item name (unicode)

Return type unicode

Returns the parent item name (or empty string for toplevel items)

MoinMoin.wikiutil.**RelItemName**(*context*, *itemname*)
Return the relative item name for some context.

Parameters

- **context** – name of the item where "itemname" appears on
- **itemname** – the absolute item name

Return type unicode

Returns the relative item name

MoinMoin.wikiutil.**anchor_name_from_text**(*text*)
Generate an anchor name from the given text. This function generates valid HTML IDs matching: [A-Za-z][A-Za-z0-9:_.-]*

Note: this transformation has a special feature: when you feed it with a valid ID/name, it will return it without modification (identity transformation).

MoinMoin.wikiutil.**clean_input**(*text*, *max_len=201*)
Clean input: replace CR, LF, TAB by whitespace delete control chars

Parameters **text** – unicode text to clean (if we get str, we decode)

Return type unicode

Returns cleaned text

MoinMoin.wikiutil.**containsConflictMarker**(*text*)
Returns true if there is a conflict marker in the text.

MoinMoin.wikiutil.**drawing2fname**(*drawing*)

MoinMoin.wikiutil.**file_headers**(*filename=None*, *content_type=None*, *content_length=None*)
Compute http headers for sending a file

Parameters

- **filename** – filename for autodetecting content_type (unicode, default: None)
- **content_type** – content-type header value (str, default: autodetect from filename)
- **content_length** – for content-length header (int, default:None)

MoinMoin.wikiutil.**getUnicodeIndexGroup**(*name*)
Return a group letter for *name*, which must be a unicode string. Currently supported: Hangul Syllables (U+AC00 - U+D7AF)

Parameters **name** – a string

Return type string

Returns group letter or None

MoinMoin.wikiutil.**get_hostname**(*addr*)
Looks up the DNS hostname for some IP address.

Parameters **addr** – IP address to look up (str)

Returns host dns name (unicode) or None (if lookup is disallowed or failed)

MoinMoin.wikiutil.**isGroupItem**(*itemname*)
Is this a name of group item?

Parameters **itemname** – the item name

Return type bool

Returns True if item is a group item

MoinMoin.wikiutil.**is_URL**(*arg, schemes=['http', 'https', 'ftp', 'file', 'mailto', 'nntp', 'news', 'ssh', 'telnet', 'irc', 'ircs', 'xmpp', 'mumble', 'webcal', 'ed2k', 'apt', 'rootz', 'gopher', 'notes', 'rtp', 'rtsp', 'rtcp']*)
Return True if arg is a URL (with a scheme given in the schemes list).

Note: there are not that many requirements for generic URLs, basically the only mandatory requirement is the ':' between scheme and rest. Scheme itself could be anything, also the rest (but we only support some schemes, as given in URI_SCHEMES, so it is a bit less ambiguous).

MoinMoin.wikiutil.**normalize_pagename**(*name*, *cfg*)
Normalize page name

Prevent creating page names with invisible characters or funny whitespace that might confuse the users or abuse the wiki, or just does not make sense.

Restrict even more group pages, so they can be used inside acl lines.

Parameters **name** – page name, unicode

Return type unicode

Returns decoded and sanitized page name

MoinMoin.wikiutil.**split_anchor**(*pagename*)

Split a pagename that (optionally) has an anchor into the real pagename and the anchor part. If there is no anchor, it returns an empty string for the anchor.

Note: if pagename contains a # (as part of the pagename, not as anchor), you can use a trick to make it work nevertheless: just append a # at the end: "C##" returns ("C#", "") "Problem #1#" returns ("Problem #1", "")

TODO: We shouldn't deal with composite pagename#anchor strings, but keep it separate. Current approach: [[pagename#anchor|label|attr=val,&qarg=qval]] Future approach: [[pagename|label|attr=val,&qarg=qval,#anchor]] The future approach will avoid problems when there is a # in the pagename part (and no anchor). Also, we need to append #anchor at the END of the generated URL (AFTER the query string).

Module contents

MoinMoin - a wiki engine in Python.

CHAPTER 9

Indices and Tables

- genindex
- modindex
- search
- glossary

Python Module Index

i

l

m

s

t

u

w

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z