Pie Web M{a,e}sher

"Life is a dung pie from which you take a bite every day"

- French proverb

What Is Pie?

Pie is just another web-based content composition and management environment driving the idea of collaborate editing and sharing, an idea that has been taken to great extends lately by a crusade called Wiki. Although so-called content management systems existed long before, employing both, proprietary as well as standardized means, the Wiki mythos pushed this effort even farther. As with XML, another internet hype, and one that is still based many illusions upon (careless marketing strategists and clueless mortals have conventionalized XML to be one of those black magic three-letter acronyms that, if adored seriously, is able to cook coffee, change your kid's nappies and, all things considered, makes you a better human being once you've sacrificed yourself fully to its course), Wikis more and more come to the public's attention and you clearly classify yourself as an outsider if you do not embrace the new technology's blessing.

Many popular software projects have much in common with economic growth: once they are unleashed upon the populace, grapping and holding their and all bystanders' attention, they can not be stopped anymore until, of course, the climax is reached and everyone turns away, bored to death.

Nevertheless, Pie poses another attempt to bore people willing to pay attention, offering its services freely and obendiently to those who are willing to accept them.

What Can It Be Used For?

Pie has been designed to be tasty and can be served with a variety of ingredients. Basically, it serves you by gladly feeding upon your mental output, this being mostly text, but also graphics, music, programs and other intellectual property.

Before you build up your illusions right away, at the very start, it should be mentioned that Pie *is unable to cook coffee*. This fact cannot be stressed and overemphasized enough. Being a content management system, Pie does lots of things for you, but it certainly can't cook coffee. If this was one of your basic requirements, you may cut off here, right now, and go for a secretary instead. (Hopefully, you didn't take this barking of your shin too seriously. Yet, it might have saved you a lot of time, if properly prepared coffee really was what you were looking for.)

Imagine, just for a moment, that you'd like to collect your thoughts, on any conceivable matter. You could handle this the conventional way by writing them down in your diary. Still, you are a modern person, always having preferred to employ technology whereever you can. Consequently, you foreclosed, of course, to make use or your PC's text processing features, thus being able to edit, cut and paste, and move around and modify your text in whatever manner and as often as you like.

Let us take this scenario one step farther: you not only want to be able to manage your thoughts at home or at work, but virtually everywhere and anytime. Sure, you can use a laptop for this purpose. However, being a social being, you want others to participate and take in (or even contribute to) your ideas - again, anytime and everywhere, as long as the participants have access to moderate 20's century technology. You may send your data and files back and forth, using e-mail, or you could even rely on more sophisicated technology, like CVS, or you could make use of your proprietary publishing software's distributed auto-update feature to spread your latest thoughts to remote clients, the participants, anywhere in the Net. Alternatively, you might rely on one of the Net's most accepted means: the Web. Doing so, not only enables you to refrain from ugly, expensive, proprietary software, but also connects you, and those with whom you intend to share your intellectual propery, using a simple, standardized means.

Tell Me More

Just guessing: you are probably not interested in any more philosophy and have already grasped the underlying idea of web-sharing documents, don't you? You would like to see the prudent facts now. To do so, you are advised to go through Pie's technical documentation. For the patient reader, there is also some text on Pie's basic concept as well as a brief summery of its features.

Basic Concepts

Many of the latest content management systems of the Wiki family bristle with sophisticated features, bells and whistles, that help to place this particular piece of software in the lead and ahead of its competitors. Some of these functions are even helpful, or at least can be considered nice to have.

Pie claims to be different. Pie has been developed and written from scratch with its primary directive being simplicity, speed and efficiency, while still being easy to comprehend (both, from a user's as well as from a developers perspective). Being easy to take in, Pie proves to be adaptable and extendable.

Interpreter vs. Compiler

Many of the latest derivatives of the Wiki clan (those that are almost able to cook coffee) lack performance in every-day use, even when run on fast web servers. The reason for this is not a lack of programming skills of its creators, of cource, but that fact that these implementations almost always interpret the sources of their pages at run-time. While this approach provides authors and users with an almost infinite amount of options of how to manipulate the output when and while being requested by a user, it certainly costs precious CPU-cycles at the very moment of request. Readers have to sit and wait.

This is where Pie tries to make a difference.

Pie does not try to cache pages it has once generated as do some of its more sophisticated cousins, but refrains from interpreting page sources at run-time at all. Instead, it compiles your page once you enter it, furthermore just conveying the precompiled output to the reader.

No doubt, this approach prevents Pie from being able to provide its users with all kinds of features that only could be offered if its pages were interpreted at run-time. It also lacks flexibility and adaptability in regard of automatically reformatting its output once alterations are made on a system-wide basis. But this is a toll Pie is gladly willing to take for the sake of efficiency.

To understand why run-time interpretation so terribly slows down the delivery of a page you have to keep in mind that it not only involves PHP-code (Pie is entirely written in PHP) to be interpreted and executed, but numerous regular expressions to take place. Due to their very nature, regular expressions take quite some time to be compiled and executed, and even fractions of microseconds easily add up to seconds and more while just a single document is being processed, regardless of the underlying code libraries' effectiveness and efficiency.

Given the WORM-scenario (write once, read many), a paraphrasis that describes the fact that pages are usually rarely changed and updated, but often read, Pie chooses to (pre-)compile its pages to be able to instantly send the result to the reader once a page is requested. Doing so, Pie tries to get rid of the run-time compilation dilemma once and for all. It's that simple.

For those who still demand a CMS to be able to provide dynamical, run-time generated output of some sort, Pie offers possibilities to tackle this task, too. Still, these features are more or less restricted to the inclusion of arbitrary PHP-code at the beginning and the end of a document. The actual page contents are static once they have been compiled.

Pie's original primary design goal was to run fluently on its creator's sedate web server and still deliver output at a considerable rate. At least on that score, it has succeeded, performing smoothly in every-day heavy-duty use.

Simplicity

Pie claims to be easy to understand. This not only covers the web-based user interface where most Wiki derivatives really do not differ that much, but also its code base. Pie's code, being (to a certain extend) highly modular, should be easy to learn, understand and extend for most programmers - which leads straight to the next topic:

Extendability

Programmers should find themselves in the position of having little or no difficulty to extend Pie in whatever direction and for whatever purpose.

Features

A brief summery of Pie's features:

- Quick page processing and short run-time latency
- Low system requirements (both, in regard of the running server's performance as well as the complexity of the installed software base)
- Independent of ODBC, SQL and other database systems, both, session-based and file-based
- Multi-user handling
- Page locking
- Page versioning
- A rather extensive user manual
- Small and simple code library
- Easily adaptable and extendable to personal needs
- Weighs less than 200 blocks (i. e. 100 KB), including online documentation
- XHTML 1.0 clean output
- Peachy

However, there are also certain drawbacks and trade offs:

- (Almost) no page-related run-time processing
- Less feature-rich than comparable Wiki-siblings
- Does not end wars (mayby), does not make love (probably), can't cook coffee (for sure)

System Requirements

Pie requires a run-time-capable version of PHP, revision 4.1.0 or later, with the following modules compiled in:

- Standard regex library
- PCRE-support

The EUID your web server is run as also requires write permission to a series of files and directories.

Software Installation

1. Create/choose the place where you want Pie to reside in. Pie's current working directory does not necessarily have to be a document root of your web server. Any HTTP-accessible sub directory will do.
2. Open `pie.php` and make all necessary changes to its configuration, especially with regard to file names and paths. Files and paths can be specified relative to the current working directory or in terms of an absolute file path.

   Although it does not pose a threat with regard to your web server's security, you probably want to move most directories out of your server's working directory. None but the file pie.php and the CSS directory have to reside in a place directly accessible via HTTP.

3. Make sure all read/write directories and files are set up in mode 0777 for your web server's run-time user to be able to have full access to create and maintain files. This applies to `pagepath`, `sourcepath`, `binarypath`, `temppath`, `lockpath` and `sessionpath` as well as to the director{y,ies} where the files `logfile` and `userfile` are located and being maintained, respectively.
4. If you use directory hashing by setting option filehash to a positive integer to represent the number of second-level directories in `pagepath`, `sourcepath`, `binarypath` and `trashpath`, you have to manually create the respective amount of sub directories in these paths, which can be easily done via

```
perl -e 'foreach $_ (0 .. HASH) { mkdir $_; chmod 0777, $_; }'
```

with HASH being `filehash` minus one.

5. Make sure to modify the administrator's password in pie.php. You can also modify the superuser's name (`admin`), if you want to.
6. Run `pie.php` and log in as administrator.
7. Access the maintenance area and fine-tune your installation. You may want to import a series of pages that are included in the standard distribution (`etc/src`), either to test your installation or to install the online documentation permanently.
8. Have fun.

Run-Time Configuration

Pie uses the global variable `$GLOBALS['pie']`, an associative array, in the file `pie.php` to hold all of its run-time configuration settings. Setting up the run-time configuration takes place before anything else happens and thus applies to all actions that are to be executed at run-time. You have to manually edit `pie.php` to change any of these values.

What follows is a list of all available options, ordered by type and name. All given names refer to the *switch*-part of the global associative array. Default values are given in brackets.

General Options

admin

Name to be used to log in as the superuser. See the `password`-entry below for the corresponding password. [`admin`]

filehash

If set, working directories for sources, pages, binaries and trashed items (see below) are stored in a second-level directory hierarchy with an amount of sub directories that equals this number. Employing hashed directories is especially useful for installations that have to deal with a large number of files and may be even necessary if thousands of files are being dealt with. If unset, all files of the same type (sources, compiled output, binaries and trashed items) are stored in one directory. Note that, if used, the sub directories must be manually created before the first files are created and stored. [0]

password

The superuser's login password is configured here. Make sure to change it after you have logged in for the first time. [`secret`]

public

If set, anyone is permitted to create a new user. This user, in turn, may now modify the system's contents. If unset, nobody is permitted to create a new user account. Registered users (those who have already registered) may still log in and do their work. To keep anyone but the superuser from logging in you would have to set this option to `false` as well as to delete the existing user database (see `userfile` below). [true]

timeformat

A format string that is passed to PHP's date()-function whenever time strings are being generated. Change this string if you do not like the output format or if it does not fit properly into your locale. [`Y-m-d H:i:s`] (ANSI-specification)

timeout

Number of seconds of inactivity after which authenticated user sessions are automatically canceled. Matter-of-factly, this gives you this amount of seconds to enter your changes for any page you want to create or edit. [3600]

version

An ID string that identifies the current software release.

Page Expiration

The following options take care of a page's expiration.

expirecount
: Number of versions that are allowed to be kept for a page. If the amount of versions ever exceeds this value, the oldest version is deleted. If set to `0`, that is, is unset, no expiration with regard of a page's history size takes place. See the section about expiration on how and when pages are being expired. [0]

expiredays
: Number of days after which expiration of a page takes place. If set to `0`, that is, is unset, no expiration with regard of a page's age takes place. See the section about expiration on how and when pages are being expired. [0]

expiresize
: Number of bytes each page file is allowed to grow up to, including all the page's former versions. After this amount of bytes is reached, Pie starts trimming older versions of the page, starting with the oldest version available. If set to `0`, that is, is unset, no expiration with regard of a page's file's size takes place. See the section about expiration on how and when pages are being expired. [67108864] (64 MB)

Options `expirecount`, `expiredays` and `expiresize` can be arbitrarily combined, but at least one version of a page, the current one, is kept at all times, regardless of the resulting file's age or size.

Pattern Matching

All strings represent patterns that are passed to PHP's PRCE-compatible pattern matching functions and that have to be matched by a user's input to be recognized as a candidate for the specific type of resource.

filepattern
: Pattern a local link has to match to be recognized as a local binary file. Default: `[[:upper:]]\w+.[0-9a-z]{1,5}`

pagepattern
: Pattern a local link has to match to be recognized as a local page link. Default: `[[:upper:]]\w+`

userpattern
: Pattern local user names have to match. Default: `\w{2,30}`

wikipattern
: Pattern tokens in continuous text have to match to be recognized as a Wiki-link. If unset, no automatic recognition for Wiki-names takes place while the output of a page is compiled. Default: `([A-Z][a-z]+){2,}`

grouppattern
: Pattern the group part of a page has to match to be recognized as a group. Default: `[[:upper:]]\w+`

maxnamelength
: Certain operating systems pose limitations on the type of characters that can be chosen for path names as well as the total length of such a character string. This option is used to limit the length of any name used to determine a locale resource, that is, a page or file name. Note that, depending on the value of option `nameencode`, the length of a file name may extend up to three times, if it consists merely or mostly of special, non-printable characters. [80]

Make sure not to include the circumflex (`^`) and dollar sign (`$`) at the beginning and end of a pattern, respectively. Pie often evaluates the specified patterns as part of and inside other patterns.

Note that, depending in the web server's capabilities to handle locales, PRCE-compatible patterns like `\w`, `[:upper:]` and `[:lower:]` may not include and correctly deal with localized or country-specific characters such as letters with accents. Depending on the configured character encoding (default: ISO 8859-1 aka Latin-1), you may have to add and specify such character ranges explicitely.

File Names & Paths

All file paths represent names of files and directories that are accessed within the run-time context of the web server's file system. They can be specified as absolute path names (starting with a slash) or path names relative to the current working directory (without a slash).

Note that all files that perform frequent updates, such as the `pagepath`, `sourcepath`, `binarypath`, `trashpath` and `tempppath` hierarchy, should reside in the same logical file system.

binarypath
: Directory to hold uploaded binary files. [bin] (R/W)

custompath
: Directory to hold customized text. [etc]

filetypes
: Text file to hold the file name extension to file type translation table. [etc/filetypes]

librarypath
: Directory to hold Pie's PHP-scripts. [lib]

localefile
: Text file to hold the locale translation table. [etc/locale]

lockpath
: Directory to hold locks for pages that are being worked on. [run/locks] (R/W)

logfile
: Text file to which all contents-modifying activity is being logged. [run/access] (R/W)

messagepath
: Directory to hold files with skeleton messages that are displayed when user interaction takes place. [etc/msg]

pagepath
: Directory to hold pre-compiled HTML-code for all pages. [out] (R/W)

sessionpath
: Directory to hold session-related data for logged-in users. [run/sessions] (R/W)

sourcepath
: Directory to hold the sources for all versions of all pages. [src] (R/W)

templatepath
: Directory that holds templates that can be used when creating new pages. [etc/templates]

temppath
: Directory to be temporarily used for file uploads as well as files that are generated at run-time. [run/tmp] (R/W)

trashpath
: Directory to hold deleted (i. e. trashed) pages. [run/trash] (R/W)

userfile
: Text file to hold user-related information, i. e usernames and passwords. [run/users] (R/W)

nameencode
: To get rid of special characters (such as slashes) that might confuse file systems, character strings that represent resource names are escaped, thereby resolving such characters into harmless letters and digits. Doing so always extends a file name's length, but you have full control over the function being employed to perform the

conversion. `rfc1738`, the default value, uses rawurlencode() (and thus a translation scheme according to RFC 1738) to convert special characters into three-character hexadecimal sequences. This works great if only a few special characters are expected to be contained in the average file name. For languages with lots of special characters and/or frequent multi-byte encoding, such as eastern Asian languages, `base64` might turn out as the better choice. This value encodes file names using base64_encode() and yields shorter lengths for file names that contain a lot of special characters.

`R/W` indicates that read/write capabilities/permissions are required for the respective file or path to function properly. Those instances have to be set up in file mode 0777 (directories) and 0666 (files), respectively. Otherwise, the web server's run-time UID won't be able to modify their contents.

Link Patterns

To suit your personal preferences of how links are being rendered use the following options to customize the way such character strings are put together. This is especially useful for sites that plan to make use of their web server's rewriting capabilities.

pagelink
String to be used for links to local pages.

filelink
String to be used for links to local binaries, that is, files that download to the user's computer.

medialink
String to be used for links to media files likes images, sounds and movies. Unlike "ordinary" files, media files are not downloaded, but embedded into the current web page.

actionlink
Action links are links to trigger actions. They are never found inside documents and there is usually no need to beautify the default settings.

There are a number of special tokens that can be used inside link patterns:

[[$0]]
Refers to the variable `$_SERVER['PHP_SELF']`.

[[$page]]
Refers to the name of the referenced page.

[[$file]]
Refers to the name of the referenced file.

[[$argv]]
Refers to the resolved string of certain values that have to be passed to the web server by means of the URL.

Example: To instruct the server to render all page links as `pie/PageName`, a format that is nice to look at as well as suitable for bookmarking and indexing by search engines, you'd set `pagelink` to `/pie/[[$page]]`. However, you would have to instruct your web server to convert requests of such URLs back a format it actually understands. As for Apache's mod_rewrite, you'd instruct it as follows:

```
<IfModule mod_rewrite.c>
RewriteEngine On
RewriteRule ^pie/(.+)$ /pie.php?page=$1
RewriteRule css/(.+)$ /css/$1
RewriteRule img/(.+)$ /img/$1
</IfModule>
```

Note that you may also have to adapt any hardcoded links available in files outside the librarypath that provide customized, static content, such as `etc/head.php`.

Local Customization

Function ...

Unless you are a software developer and want to modify Pie's principal behaviour, you are probably not interesed in the files in `librarypath`. To modify low-level functions, have a look at the files `common.php` and `toolbox.php`, respectively. They contain basic helper functions most other high-level functions are being based upon.

... and Form

Apart from `pie.php`, which, acting as a master multiplexer, holds the system's central run-time configuration, you primarily do your local customization of your site setup in static, read-only files that reside in `custompath`.

All HTML output that is being rendered at run-time consists of a header (`custompath/head.php`), the actual page's contents, and a footer/trailer (`custompath/tail.php`). You can modify the header and trailer in any way that suits your needs.

To change the system's language, edit the file specified via option `localefile`.

Your site's overall graphical appearance is customized via CSS. For instance, links to internal pages, internal binaries, external resources and e-mail addresses are all assigned their unique classes, respectively, and each can be rendered in a different fashion, if need be. Unlike most other Pie components, your site's CSS-directory must reside somewhere below the web server's document root. Otherwise, the browsers' HTTP-requests won't be able to retrieve its contents. You typically import CSS-directives via second-stage HTTP-requests through your header files (like `head.php` in `custompath`), but could also set up the style somewhere within the header file itself.

Registration & Login

Registration

While everybody can browse and display *all* the server's pages and files, only registered users can create and modify pages and upload new files.

To create a new user account, click on **Login**, and then, instead of actually logging in, click on **Register**. This causes a registration form to appear, asking you for a username and a password for the new user.

The username must match the variable userpattern to be accepted. The password has to be entered twice to reduce the chance of accidental typing errors.

After registering, you can log into the server using your new username and password.

Login

Using separate user accounts for different persons allows for a playground of multiple identities. Once you have successfully logged into the server, you can modify its contents, including creating, modifying and deleting pages as well as uploading files.

If the public-switch is set to `false`, new users are not permitted to register, but existing users can still log into the system.

Creating Pages

One of the frequently asked questions of new users not accustomed to the Wiki-way is: "*Where is the button, link or gadget to create new pages?*"
Usually, the oh-so-simple answer is: "*There is none. You don't click buttons to summon pages out of thin air.*" Well, actually you could just do that, but it's not what the Wiki-way is all about.

Typically, you don't create new pages by themselves. Doing so would inevitably result in the new page being an *orphaned* page, that is, a parentless page that is not referred by an other page. In lieu thereof, you rather link pages *from* other, already existing, pages. Linking pages via so-called hyper text links is a central dogma of Wiki-based software, much the same as it is in World Wide Web.

The Top-Down Approach

To create a new page, you simply place a link to it on an other page. As for Pie, there is always at least one single page: the home page, also called MainPage. It is the root document of all other pages and parent to any number of children and descendants. Theoretically, the MainPage is the only orphaned page, but even the MainPage may be referred and pointed back at by any of its children or grand children, thereby freeing it from its orphaned state.

To place a link on one of your pages, you edit the page and type in the link. Links are character strings that are enclosed in double brackets, the string representing the name of the page your are referring to. Once the parent page has been updated and compiled, it features a clickable link to its new (but still missing) child.

We call the new page MyPage and would now like to create and edit it, so we place the string `[[MyPage]]` somewhere in the text of the parent page. When you update the parent page, display its content and click on the new link, Pie will confront you with a message:

```
Page MyPage could not be found.
To create this page, click here.
```

Boom, there it is! - This is the button to create your page! When clicked, a form to compose the new page follows right ahead. If your server offers templates, you can choose one to get an overview of the layout and structure of the newly created page-to-be.

That's the way it works. That's the way you create pages. And that's the way you make sure not to generate orphaned pages.

Deleting & Renaming Pages

To delete a page, either click **Delete** in the option panel or display the candidate and click **Edit**. Now remove all characters from the editing field. Once you click **Update** you will be asked whether or not to completely delete the page from the server, since empty pages are not allowed around.

Deleting and renaming pages is dangerous because local and remote links do not get updated automatically, but simply stop working.

Page Names

Wiki Names

One way to name pages is to use Wiki-names. Wiki-names consist of upper and lower case characters only. The first character of a Wiki-name must be an upper case character, followed by one or more lower case characters. Somewhere after the second character of the name, another upper case character has to follow, which, in turn, is followed by lower case characters. You need at least two upper case characters to compose a valid Wiki-name with an arbitrary amount of lower case characters on the right side of each of them, respectively.

- Valid Wiki-names: `MyPage`, `RisingSun`, `NotToBeForgotten`.
- Invalid Wiki-names: `mypage`, `myPage`, `Mypage`.

Pie Names

Pie is usually configured to be less picky about the combination of characters it accepts for page names. Its default behaviour is to be happy with any a name that starts with a capital letter followed by any combination of letters, digits and underscores.

The actual format for page names, that is, the pattern names have to match to be considered valid names for pages and files, is configured using the respective variables of the server configuration.

Linking Pages

Links Inside Continuous Text

The simplest way of linking to resources is to make use of Pie's auto recognition of certain types of character strings just as you type them.

Wiki-Names

The first link type Pie regognizes is character strings consisting of strict, native Wiki-names. Wiki-names placed somewhere in continuous text are automatically recognized as Wiki-links, that is, links to other pages on the same server.

The switch wikipattern is responsible to decide whether or not this feature is enabled and, if so, to which pattern it applies.

External Links

The second type Pie regognizes is character strings consisting of valid URLs. URLs, too, are displayed as links embedded into the surrounding continuous text. Note that a valid URL needs its protocol to be included. Consequently, http://www.gnu.org/ is recognized as a link, but www.gnu.org is not. Links referring to external resources are always opened in a new window.

The third type of recognized links is e-mail addresses. If your text contains a valid e-mail address, like foo@bar.baz, it should be recognized as such and rendered as an e-mail link instead of static continuous text, enabling readers to click it to address the referenced recipient.

You can always put double brackets around links to external resources to make their meaning stand out more clearly. For weird addresses, this may sometimes be the only way to force them to be recognized as links at all.

Internal Links

In the section about Creating Pages you learn why and how to create links to new or already existing pages. To put an internal link onto your page, enclose its name into double brackets: `[[MyPage]]` creates a link to the homonymous page.

Wiki-names, when enabled and as long as in their native format, are automatically recognized as links when you type them and do not need to be enclosed in double brackets. It is a question of personal style whether or not to provide the additional brackets on both sides of the Wiki-word. Some people don't mind typing the four extra key strokes to make their sources more readable and bullet proof, while others claim that this would contradict the Wiki-way.

Internal Anchors

You can place anchors anywhere in your document using the `anchor:` selector followed by the label anywhere in your text. The label should consist of a character string that abides the HTML standard. Labeled page sections can be addressed from other pages via links like `PageName#LabelName`.

Alternate Text

To provide an alternate text to be displayed instead of your link, append the alternate text after a blank or vertical bar (`|`) after the actual link. Once you have chosen the bar as the content separator, you can use blanks on either side of it, whereas the absence of a bar automatically selects blanks to serve as separators. The order of these components is always the same, regardless of whether you address pages, files or external resources:

```
[[Link or resource name|Options (if any)|Alternate Text]]
```

```
This is [[MyPage|my personal page]].
```

```
Please refer to the [[http://www.gnu.org/licenses/gpl.html GPL]].
```

This creates the two indicated links, but prints the alternate text instead of the page name and URL, respectively.

Page Groups

(Wiki) groups are a method of classifying pages (or files) into groups of related pages. Links represent pointers between pages of the same group just as usual. Such a link is called a *relative path* (or *relative addressing scheme*) since it implies the presence of the group name, which is the same for the source and the destination page.

To create a link to a page in another group, you have to specify the group name as well as the page name and separate both with a slash, thus yielding the *absolute name* of the destination. This is called an *absolute addressing scheme* because the entire path is part of the link name. Internally, all pages are stored using their absolute names; relative addressing just offers a convenient and comfortable way to refer to contents of a similar context.

There is one special group, the *root group*, which is asumed if there is no group context available. The root document of your server, usually called `MainPage`, resides in the root group, for example. To address a page of the root group from a page of another group you have to put a slash as the first character of the referring link, like `/MainPage`.

```
Neighborhood: [[SiblingPage]]
Different context: [[OtherGroup/PageName]]
Root group: [[/MainPage]]
Files: [file:PrivateFiles/CurriculumVitae.pdf|My resume]
```

Files, Media Files and Images

To embed the contents of files in your documents you have to transfer them to the web server independently from the page that is supposed to link to them. You can not compose binaries by the means provided by Pie, but have to create them offline and upload them en bloc. Still, the top-down approach of content creation also applies to files. As with pages, where a parent introduces a child through a link, files, too, can be brought into existence by placing links on a page and by following such a link to the respective upload form.

As with pages, you should assign your files significant and meaningful names. Unlike page names (and unless specified otherwise), the name of a file ends with a period, followed by an extension that determines its type much the same as it is the case with most file system standards. `MyFile.zip` and `MyDocument.pdf` are examples for file names with file type extensions. This requirement regarding the syntax of file names apply to all types of files, media files and images.

Linking Files

Files, regardless of their actual content, are linked by the `file:` selector. `[file:UserManual.pdf]` represents a link to the local file `UserManual.pdf`. Such links, when being clicked, cause the respective file to be downloaded to the user's computer. If the linked file does not exist (yet), an upload form appears instead, asking the user to upload the specified file. You can verify the links of a page to make sure all its links (to pages and files) point to valid destinations, and, if the linked files do not exist, upload them one by one.

Note that all images and media files can be treated as ordinary, downloadble files by employing the `file:` selector:

```
 [file:ImageFile.tiff Click here] to download this image as a file.
```

Media Files (Embedded Files)

Media files, like graphics, sound and movies, are a special kind of files and are usually treated differently in that they are embedded into your documents instead of being just linked as downloadable resources. Media files are linked via the `embed:` or the `media:` selector. Unlike files and images, media files support no alternate text, but interprete any additional text, if any, as their display parameters:

```
 [embed:MovieClip.mpg width="320" height="240" controller="true"]
```

Images

The last type of files are images. Like media files, images are not downloaded, but rendered and displayed as part of the content of a page. Images are linked using the `image:` selector with the name of the refered file as its argument.

```
[image:MyPortrait.jpg That's me!]
[image:CanyonView.jpg|The Grand Canyon at sunset]
```

If a `class:` selector follows the `image:` selector and the file name, the specified class is applied to that image. This is mostly useful for design purposes, positioning or bending images in a special way.

Note that the referenced class must be declared and available somewhere in the web server's CSS-file.

```
[image:ProductView.jpg class:floatRight Brand new model]
```

Links, images, classes and alternate text can be combined in this very order. To create a link to a local page or external resource in form of a clickable image whose alternate text you also like to determine, you would say:

```
[[UserPages/FooBar image:UserPages/FooBar/PortraitPhoto.jpg Follow me]]
[[http://host.domain.com/ image:LocalImage.png Click this link]]
```

Editing Pages

The Interface

Once you have decided to create a new page or to edit an existing one, a form with a textarea shows up. The textarea is either empty (for new pages) or filled with the source code of the specified page. If your server has customized templates installed, you can select one of them to use it for the overall layout of your new page. In this case, the content of the template is displayed in the textarea.

Use your keyboard's cursor keys to navigate the textarea and perform the usual word processing supported by your browser. Make use of your mouse and your operating system's capabilities to copy, cut and paste chunks of text.

The latest versions of Pie also feature a minimalistic JavaScript-based graphical user interface to summon some of the more frequently used formatting rules. The buttons of the toolbar apply to a marked section of the textarea, respectively, so you first mark the text chunk you would like the operation to apply to and then click the button that triggers it. If you do not mark any text beforehand, you are prompted for the text, which, along with the necessary formatting, is then inserted into the textarea.

You can enter an optional text as a comment for the page you are editing. If you work on a page whose latest version was also edited by you, you can tick the **Minor update** option, meaning that instead of creating a new version of the current page the existing version is supposed to be replaced. This is useful for subsequent updates that merely correct a number of spelling mistakes, but neither add important content, nor perform major changes to the structure of the page.

If you would like to cancel editing (or creating) a page, make sure to click **Cancel** instead of just abandoning the editor. By starting to edit a page, you implicitely set a lock to it to keep it from being modified by other users at the same time. Clicking **Cancel** removes this lock and marks the page as being available for other editors.

Expiration

Expiration of a page takes place in the background while a page is updated and transferred back to the server. Expiration requires the respective values to be configured in the server settings:

If `expirecount` is set to a positive integer value, the number of held versions of an updated page will not exceed the specified value, with older versions being purged and not available anymore.

If `expiredays` is set to a positive integer value, available page versions will not grow older than the specified number of days, older versions being purged and not available anymore.

You are free to combine `expirecount` and `expiredays` in any way you want. Still, at least one valid version of a page, the current version, is kept at any time.

Expiration does not take place event-triggered or periodically, but just when pages are being worked on and the database files are updated. Even then, only former versions of the updated page are expired. All other pages are left untouched.

Notes

You might want to look at the source code of other pages to become familiar of how to use commands and format your text.

Pie's various options to format text slightly differs from the syntax used by other Wikis. You should take this into account when moving in from other implementations and start editing your texts with Pie.

Markup Code

Pie offers authors numerous options to format their text as they type it into the web form:

Lines and Paragraphs

Multiple paragraphs are marked as such by separating any two of them by an empty line:

```
First paragraph

Second paragraph
```

Two lines that are not separated by an empty line but a single line break only are treated exactly like this: a line break is forced at the end of the first line and the second line starts afresh.

```
First line.
Second line.
```

You can surpress a line break at the end of the first line by placing a backslash (\) at the end of that line, folding the two successive lines together.

```
The quick brown fox \
jumps over the lazy dog.
```

Environmental Contexts

The content of a line can be treated in a special way by placing selected characters as the very first character at the beginning of that line. Depending on this character, called the selector, special formatting of this line takes place. You may put any number of blank spaces between the selector and the rest of the line to to make it stand out more clearly.

Selectors that are placed in front of subsequent lines and that are "compatible" with each other are considered to be part of the same logical, environmental context. Such a context can be thought of as an compound of multiple lines, each of which is supposed to be treated similarly. Two lines separated by an empty line, and thus creating two paragraphs, always break environmental contexts.

Verbatim Output

An exclaimation mark selector tells Pie to treat the line as verbatim output, that is, to be displayed *as is* with no further formatting applied.

```
!#!/bin/sh
!while [ 0 ] ; do
!    echo Ping
!    sleep 1
!done
```

renders this very script (without the leading exclaimation marks).

Comments

A semicolon introduces a comment. The rest of the line is simply skipped and not included in the output.

```
; Begin of header
Greetings stranger
; Begin of body text
[...]
```

A line with two semicolons prepended is also treated as a comment. Unlike a single semicolon, however, the contents of the line is still included in the output, yet as a hidden comment instead of readable text.

```
;; I am a comment
```

translates to the invisible (yet existing) HTML output of

```
<!-- I am comment -->
```

Comments of both types do not break environmental contexts.

Separators

Three dashes (minus signs) put together as the only characters of a line create a horizontal separator and can be used to structure and separate multiple parts of a document.

Paragraph Alignment

You can align the lines of a paragraph to the left, right, center as well as to justified output, respectively, by calling upon a <, >, >< or <>-selector, respectively.

Note that you have to break the contextual environment of a paragraph to alter the alignment of its contents altogether:

```
>< This line is centered.
> Although not lead by "><",
< the rest of this paragraph is also centered
because it belongs to the same environmental context.
```

yields

This line is centered.
Although not lead by a "><",
the rest of this paragraph is also centered
because it belongs to the same environmental context.

Headlines and Titles

To create the HTML-entities `h1`, `h2`, etc., you put a respective amount of =-signs at the beginning of its line. You can go down as far as five ='s to specify the fifth level of subtitles.

```
= Main Title
I am the preamble
== Sub Title
In this section you learn to format your text.
=== Topic One
=== Topic Two
[...]
```

Headlines always break environmental contexts, creating miniature contexts of their own. You may insert blank lines around them to make the section they introduce stand out more clearly.

Tables

Tables can be rendered via the vertical-bar-selector: A "|" put as the first character of a line marks it as part of a table environment. The columns of a table row are separated from one another via further vertical bars, as is the end of the line to terminate the table row. All table rows must contain the same number of columns.

Per default, the contents of all table cells are aligned as determined either by the browser's settings or the currently applying CSS. To explicitely declare the contents of a cell to be horizontally aligned, place > (right), < (left), >< (centered) or <> (justified) selectors as the first characters of the respective cell.

If an =-selector is put as either the very first character of a cell, or the first character after a cell's alignment specification, this cell is treated as part of a table header instead of an ordinary table row.

```
|=First Column|=Second Column|=Third Column|
|alpha        |beta          |gamma|
|<left |>< centered |> right|
```

translates to

First Column	Second Column	Third Column
alpha	beta	gamma
left	centered	right

Escaping Selectors

You can escape the first character (and potential selector) of a line by placing a backslash (\) in front of all other characters. Doing so ignores the selector and treats it as a literal character.

A backslash also escapes any special character that might appear somewhere in your text, enabling you to literally use such characters in your text:

```
This is not a link, but just bracketed text: \[\[Text\]\].
```

Remember also that a backslash at the end of a line prevents this line and the next from being broken apart and actually treats the two lines as if the line break was not present at all.

Lists

Lists, foremost representatives of environmental contexts, are special insofar as their items can be nested within one another, the indent of a particular list item depending on the depth of the surrounding environment. Lists are either terminated by an empty line or any construct that forces an environment to end prematurely, as it is the case with headlines and horizontal separators.

Ordered and Unordered Lists

You start unordered, "bulleted" lists with an asterix (`*`) at the beginning of a line. The number of `*`-signs (or the number of blanks placed left to them) determines the amount and level of indentation at the left side of the generated list item.

Ordered lists, that is, lists that continuously increase their items' counters, start with one or more plus-signs (+) at the beginning of a line.

```
* Master item
** First sub item
 * Second sub item
* Master two
++ Count alpha
 + Count beta
* Master three
```

translates to

- Master item
    - First sub item
    - Second sub item
- Master two

1. Count alpha
2. Count beta

- Master three

Definition Lists

Definition lists (lists that are made up of items consisting of a definition term and a definition text each) are created by a colon at the beginning of a line, followed by the term, a double colon, and the text.

```
:First::First item of the list
:Second::Second item of the list
**Alpha:First sub item
**Beta:Second sub item
:Third::The third item is a [[http://pie.ekkaia.org/ link]].
```

becomes

First
: First item of the list

Second
: Second item of the list

- Alpha: First sub item
- Beta: First sub item

Third
: The third item is a link.

Note the double colon instead of a single colon to separate a term from its definition. Note also that there is only a single colon at the beginning of the line, regardless of its indent or placement inside other list items of different kinds and levels.

Text Spans

If a string of characters is enclosed in single brackets and the first character or word of this block turns out to be a selector, this block is treated as a miniature environmental context, called a *span*. Unlike environmental contexts, which comprise entire lines, spans only apply to the text chunk contained between the opening and closing brackets.

Spans have the form

```
[<selector><content>]
```

where `<selector>` is either a single special character, like `!` and `%`, or a liternal word terminated by a colon (`:`), like in `class:`. In the latter case, the colon is immediately followed by an additional argument that specifies the selector, whereas special character selectors just stand for themselves. `<content>` represents the text part the selector applies to. In case of a literal selector, you have to insert a blank between the selector and the content to mark the former's end.

Spans can be placed in continuous text as well as nested inside environmental contexts, like in lists and tables. Two or more spans must not be nested inside one another.

Verbatim Output

Just like an exclaimation mark at the beginning of a line marks verbatim output, an exclaimation mark selector in a span calls upon the span's content to be rendered as verbatim text. As for HTML, this results in the content to be enclosed in `<pre>`-tags.

To omit the surrounding `<pre>`-tags altogether use the equal sign selector (=) instead. This allows arbitrary text to be passed through to the reader unescaped.

Emphasizing Text

An underscore (`_`) selector marks the content to be underlined. An asterix (`*`) renders the content in **bold** (`<b>`) characters, while a slash (`/`) renders it in *italics*.

However, the preferred way to emphasize content is to wrap the text in shortcuts, that is, character sequences, notably `_`, `*`, `''` and `'''`. Unlike the `_`-, `*`- and `/`-selectors, shortcuts provide no formatting instructions, but rather instantiate two levels of emphasis: Text blocks (not spans!) wrapped in plain underscores or double apostrophies, `''like so'' or _so_`, are rendered in *emphasized character mode*, while text blocks wrapped in asterixes or triple apostrophes, `'''like so''' or *so*`, are rendered in **strongly emphasized character mode**.

Depending on your web browser and CSS-settings, the two modes of emphasizing text may or may not look very much alike. However, please note that from an editor's point of view, applying characteristics such as design information to content (using the selectors) is usually considered a bad style. This is a task better left to designers, while editors maintain the contents only.

```
[_This] and [/that] is not quite the same as _this_ or ''that''.
[*This] is not quite the same as *this* or '''that'''.
```

Note also that, in the case of an empty span, the underscore selector does not apply, but functions as a non-breaking space for the adjecent text (see below).

Citations

Citations are triggered via a quotation mark (double quote) selector.

```
["Believe nothing and be on your guard against everything]
```

Monospaced

To mark a span as code (which browsers usually display using a monospaced font), use an at-sign (`@`) selector.

```
Keep [@/etc/passwd] secret!
```

Size Variation

A plus and minus sign call upon the content to be rendered enlarged and shrunk, respectively.

```
This is [-small], whereas this is [+big].
```

Superscript and Subscript

Apostrophe and comma signs call upon the content to be rendered as superscript and subscript, respectively.

```
a['2] = b['2] + c['2]
p[,n] = (x[,n], y[,n], z[,n])
```

$a^2 = b^2 + c^2$
$p_n = (x_n, y_n, z_n)$

Special Characters

You can embed character into your document for which you know its HTML- or Unicode sequence.

HTML-sequences are created by employing an ampersand (`&`)-selector:

```
[&yuml], [&acirc], [&eth], [&THORN]
```

yields

ÿ, â, ð, Þ

To make use of the numerical code of special characters append a pound sign to the ampersand (`&#`) for decimal numbers. For hexadecimal numbers, an additional `x` is required.

```
A is [&64;], the colon is "[&58]".
Beyond ASCII: [&#8734;] (infinity)
Greek: [&#963;][&#959;][&#966;][&#972;][&#962;]
```

yields

A is A, the colon is ":".
Beyond ASCII: ∞; (infinity)
Greek: σ;ο;φ;ό;ς;

To include the respective characters directly into the document instead of copying their representations use the percent (`%`)-selector. A percent selector followed by a decimal number includes this number's ASCII character into the text. For hexadecimal numbers, prepend an "x".

```
A is [%65], the colon is "[%x3A]".
```

All special characters created by their names and codes, respectively, may, but need not, be terminated by a semicolon.

A character that is frequently used is the non-breaking space. It serves as a non-breakable white space between a person's title and name, or is used to separate numbers and dimensions. The NBSP is created with the special character combination `[_]` or `[ ]`.

```
Dr.[_]Leonard Horatio McCoy,[ ]M.D
```

Note that in this case the underscore (which is followed by nothing but the closing bracket) does not serve as a selector and thus does not introduce a text span.

Text Anchors

The `anchor:` selector does not render its content in a special way, but causes an internal anchor to be placed at the very position it appears. Such anchors, which consist of the selector followed by a label, can be pointed at by links within the scope of the same page, or even across page boundaries.

```
[anchor:MyLabel]
...
An important note can be found [[SpecialCase#ReadThis here]].
...
[[#MyLabel Back to top]]
```

Customized Environments

To apply a number of handcrafted CSS-rules or class definitions to a specific section of your text you can create a customized environment. The start of an environment is introduced by the `begin:div`-selector and ends with the `end:div`-selector. All content between the beginning and end mark of the environment is rendered with the specified rules applied.

Such rules usually come in form of a `.class`, or `class:`-statement, respectively:

```
[begin:div class="warning"]
; or [begin:div .warning]
; or [begin:div class:warning]
My customized content...
[end:div]
```

Named `div`s are used much less frequently:

```
[begin:div id:FooBar]
; or [begin:div #FooBar]
```

```
Content within the named div "FooBar".
[end:div]
```

Note that all referred classes and named `div`s are to be declared in the server's CSS-file. However, local rules can also be expressed by means of literal HTML-statements without calling upon predefined declarations:

```
[begin:div style="color: red; font-weight: bold;"]
More customized content.
[end:div]
```

Customized environments can be nested, but must suffice the rules of well-formed XML-definitions.

Using Selectors

Selectors are used to state temporary rules that only apply to small part of the content of a page. The most common way to employ a selector is to put it right after the opening bracket of a text span, like in

```
[class:myClass this text is classified by myClass]
```

In this case, the selector's properties only apply to the text span inside the bracket boundaries. Any content outside the brackets is left untouched.

In environmental contexts, formatting rules, like the one above, apply to the whole line in which they appear. Examples are the asterix (`*`) and plus (`+`) selectors to introduce list items or the equal sign selector (=) to evoke a header line. Such selectors are not explained here, but dealt with in the markup reference.

Selectors are also used to evoke a special functionality, like in `[file:MyFile.txt]` or `[anchor:MyAnchor]`, with the result of the triggered capability being inserted at the selector's position.

Literal Selectors

Literal selectors are made up of letters only and are always terminated by a colon (`:`). They usually appear as "bracket selectors", that is, operators that follow an opening bracket.

page:
: The argument following this selector is treated as a link to a local page. Strictly speaking, names enclosed in double brackets are a convenient abbreviated version for the `page:` and `link:` selectors, respectively.

file:
: The argument following this selector is treated as a link to a local file.

link:
: Treat the argument as a link to an external resource.

image:
: Treat the argument as a link to a local image.

embed:
: Treat the argument as a link to a media file.

media:
: Same as `embed:`.

mail:
: The argument following this selector is treated as a link to an e-mail address.

mailto:
: Same as `mail:`.

anchor:

Treat the argument as an anchor to be placed on the current page at the position where it occurs.

class:

The argument following this selector is treated as a CSS class to be applied to the text span it leads.

Single Character Selectors

/ (slash)

Render the text span in *italics*.

*** (asterix)**

Render the text span as **bold** text.

_ (underscore)

Render the text span as underlined text.

= (equal)

Treat the text span as literal content.

! (exclaimation)

Render the content in `monospaced` block mode.

@ (at)

Render the content in `monospaced` inline mode.

" (inverted commas)

Treat the text span as a *citation*.

+ (plus)

Enlarge the font size of the content of the span.

- (minus)

Shrink the font size of the content of the span.

, (comma)

$_{\text{Subscript}}$ mode.

' (apostrophe)

$^{\text{Superscript}}$ mode.

. (period)

Same as `class:`.

& (ampersand)

Insert special characters.

Displaying Page Details

This options displays a brief overview of the status of a page.

The status information includes the name of the user who modified the latest version of the page you are examining, the date when this modification took place, the comment (if available) that was saved along with this version, the page's size and length, the number of links to internal resources as well as its History size, that is, the number of changes the page has undergone so far.

If the page contains any links to internal resources, you can follow the given link to verify the availability of each of the page's links.

If the page has been updated more than once, you can follow the given link to browse the list of versions of the page.

Verifying Links

After creating or updating the source of a page that contains links, you may want to verify if these links actually point to available resources. If you create links to new pages, these descendants do not exist and you might appreciate a methodical means to create and edit all of them, one after the other.

When **Verifying** a page, the server lists all internal links of a page, if any, along with its type (i. e. a page or a binary file) and availability. For links that refer to resources that do not exist (yet) you can click and update all of the descendants until the original page's verification results in available links exclusively.

Page Versioning

Unless you decide to perform a Minor update only, which, in turn, is only possible if you also performed the latest update of the page, a modified page source does not replace a previous one, but is saved as a new version of this page. Each version of a page can be referred separately. The list of versions that have been saved for a page is called the History of a page and each entry is identified by its time stamp, the creation date of the respective version. Browsing the History from past to present allows you to experience the temporal development of page.

The History of a page is usually displayed by clicking the homonymous link in the option panel. The entries of a page's history are displayed in order of its available versions' time stamps. Along with the time stamp, each version's size, author and comment (if any) are listed. By clicking the time stamp, comment and source field, you can display each version's output, detailed page information and source code, respectively.

If you want to recycle an old version of a page, click on **Revert to** in its History table. This causes a brand new version of the page to be created, consisting of the source of the selected former version. Note that, in doing so, neither the current version, nor the former version is overwritten (unless you are the author of the latest version and decide to perform a Minor update). Instead, the new version is considered to be completely inpendent of previous versions and you can edit and make your changes just as with any other page update.

Listing Contents

Listing the contents of your server allows you to browse all user-contributed data stored and managed by Pie. You can select beween available pages, orphaned pages, Trashed Items and Binary Files.

The listing of each category includes all entries of the respective type along with the name of the resource, the date of its latest modification as well as the respective file's size in the file system. Per default, this table is limited to 30 rows at a time, but you can assign the URL's `limit` parameter any other value. Specifying an empty limit disables limiting the output.

The table can be sorted by name, modification date and size by clicking on the respective column's header. Clicking the header a second time reverses the sort order.

Searching Contents

To search pages for character strings, use the small text field labeled **Find**. Enter the text you are looking for and click the button. If one of the server's pages contains the specified character string, the respective page is listed in the forthcoming overview.

For a more advanced search mechanism that offers various options to be toggled and combined, click **Find** with an empty query string. This will evoke an advanced search user interface.

Quick Search
: This default search mode dives through the pages to find your text exactly as you type it. A page has to literally contain this character string to produce a positive match. Quick Search does not distinguish between uppercase and lowercase letters. Use this mode whenever you know exactly what you are looking for. Due to its high performance, it represents the default search mode. If disabled, the query is performed more thoroughly, also picking and listing strings with variants that merely differ in the case of the provided character string. This, however, is bought with slightly slower search performance, since pattern matching with regular expression is employed instead of literal search methods.

Match Whole Word Only
: When you activate this option, the specified search string has to stand out as a single word, that is, it has to be surounded by blanks or non-word characters or be located at the beginning or end of a sentence, respectively. Doing so, will match `king` only when this token stands for itself, like in "king and queen", but not in the middle of other words, like in "clicking".

In either mode, the server goes through the current version of each page's source code to find the respective search string. The search mechanism makes no difference between normal text and meta text, the latter being text that is not displayed on the screen when displaying a page. If, for example, you searched for pages containing a literal opening or closing bracket and, which is quite likely, had pages available with brackets on them, these pages would show up among the candidates of pages containing the brackets although the brackets are not displayed on the compiled page. In general, such meta text usually consists of control sequences like those used for selectors at the beginning of lines and spans, as well as links to other pages placed inside brackets.

Maintenance Overview

Maintaining your installation may become preferable or even necessary as soon as the number of created pages and registered users exceeds a certain limit. Maintenance is reserved to a single user, the administrator, though this burden may be shared by several people who simultaneously carry out the maintainer's duties.

Additional to being able to create and modify pages and files, administrators may run a series of maintenance commands, such as actually erase deleted pages from the system, that normal users are just not permitted to do.

To perform the special maintenance duties you have to log in as the administrator, whose username and password are specified in the system configuration.

Displaying System Information

This option provides the system administrator with a basic overview of the system's current configuration as configured in the server settings. It lists all keys and values of $GLOBALS['pie'], except the one for the administrator's password.

Page Import

This command allows an administrator to import an arbitrary amount of local files as pages. Two file formats are recognized:

1. By default, the file's name is taken as the page's name and the file's time stamp is treated as the source's last modifying date. The file's name must match the system's page pattern, otherwise, the file is skipped and not imported.
2. Files found to contain content consisting of Pie's extended source file format allows you to include further options in the file's header. It also allows you to import several versions of the same page.

The *Extended Source File Format* looks like this:

```
option1=value1
option2=value2
[...]
optionN=valueN

[source code]
```

A single empty line between the header and the source is mandatory.
Supported options are as follows (the options' order doesn't matter):

page
: The name of the page.

stamp
: Time stamp in seconds since epoch.

size
: The size of the source in bytes.

user
: Name of the user who created the source.

comment
: A comment that is to be saved along with the source.

Of these options, only `stamp` and `size` are mandatory for all files written in Extended Source Format.

All imported files must reside within the same directory, but are stored using the current installation's configuration.

Synchronization

Entered sources have to be compiled to generate the final, viewable documents. The process of compilation usually takes place as soon as you create or modify a page. Still, an administrator might want to rebuild all pages of the system at once, for instance, when page sources are imported from an other system.

Unlike editing pages, where each page is locked and unlocked to assure that only one author writes to the same page at the same time, the process of synchronization deals with all pages, one by one, without locking and unlocking them. This is why synchronization should take place rarely and only if other users are known not to write to the local pages.

Moreover, page synchronization is expensive with regard to the resources it demands from the underlying server. If a large number of local pages is at hand and/or the employed server hardware lacks the necessary speed to deal with the compilation, it may take quite some time to compile all of them in one pass. The web server might even respond with a time out because the process of compilation exceeds the maximum execution time of the running PHP process.

You can synchronize individual pages by specifying the argument `page` in the URL execution string.

System Activity

Display The Log File

This option displays a variable amount of lines at the end of the log file, a file that keeps track of all actions that modify the contents of the system, be it pages, binary files or maintenance commands.

Purge The Log File

This option trims the log file after saving a copy in $GLOBALS['pie']['temppath'] where it rests until the contents of this directory are expired.

Emptying The Trash

Pages that have been deleted by users are not directly removed from the system, but instead put into the Trash, a temporary dumpster and directory in the file system from where they can be recycled.

Recycling a page either happens explicitely by evoking the respective command, or by creating a new page with the name of a page found in the Trash. While trashed items can be normally listed by anyone, only the system administrator can empty the Trash, thereby irrecoverably erasing the trashed pages once and for all.

Behaviour At Run-time

At run-time, the stock configuration behaves as follows:

1. Unless binary output is requested and relayed to the user, `custompath/head.php` is prepended at the top of each page. This file may contain any customized code, both for static as well as dynamic page rendering. It may also be empty or absent. In the latter case, Pie will just put some minimalistic standard code at the beginning of each page.
2. Following the header comes the actual output of each page, that is, the contents of the respective precompiled page source.
3. The content of all pages is eventually followed by the content of `custompath/tail.php`, if available. If unavailable, closing tags is all that follows the preceeding output.

See chapter Run-Time Configuration for various semi-hardcoded configuration options.

Cascading Style Sheets

One of the central dogmas of Wikis is to separate content and design: "*Form follows function,*" it is said. To reach this goal, Pie makes extensive use of Cascading Style Sheets. Many of the rendered output components refer to CSS-classes, which, if not customized on the web server, appear in the standard form specified in the user's web browser. These classes allow for versatile customization to reflect the preferences associated with a web site.

List of standard classes:

pageLink
class for links to internal pages.

fileLink
class for links to local files.

mailLink
class for `mailto:`-links.

externalLink
class for links to external resources.

Editors can also call upon classes that only apply to a single text span using the `class:` selector. The `class:` selector can also be used for images where it determines the appearance and layout of a single image.

As with all the standard classes listed above, the respective statements have to be declared on the web server's CSS-file to take effect.

Advanced User Interaction

The most convenient way to interact with Pie is to click various buttons of the graphical web user interface and follow the links it offers at times. However, being highly customizable, the appearance and number of the range of buttons and links mostly depends on the local configuration. They can be added and left off as suits the needs of the local web site.

What always and everywhere remains the same (within the same software release) is the ability to directly interact with Pie by means of passing handcraftedly tailored URLs to the multiplexer, that is, the file `pie.php`. Although the multiplexer supports quite a number of arguments it follows a simple and straight scheme to evaluate its input.

The multiplexer basically supports three variables: `action`, `page` and `file`. `page` and `file` specify the *object* the multiplexer is supposed to interact with, while `action` tells it what actually to do with this object. If you want Pie to create a new page for you, you would tell it just that:

```
http://my.host.domain/path/pie.php?action=create&page=PageName
```

Sometimes, additional parameters are required to trigger the demanded action, like to rename pages or files which require a source name as well as a destination, but most basic capabilities only use the `action`, `page` and `file` parameters, respectively, the latter two comprising the types of resources Pie knows how to deal with.

What follows is a list of all available, externally executable actions, that is, values the `action` parameter can assume. Some actions are supposed to be called directly, while others are called upon by subsequent user interaction of previous actions. Some require a page to be specified (demanding a `page` parameter), while others work only on files. Yet others can be used to operate on either object type.

Public Mode

The public (read-only) mode is used to browse the site, display contents and download files.

display
: Used to display the content of a page. It is used implicitely, if no action is specified. It operates on pages.

dump
: Used to push the content of a file down to the user's browser. It operates on files.

download
: Same as `dump`, but causes the file to be saved to a file on the user's computer instead of being interpreted by the browser.

info
: Used to display details about the current version of a page. Operates on pages.

history
: Displays a list of former versions of a page. Operates on pages.

source
: Displays the source code of a page. Operates on pages.

verify
: Displays a list of all local links of a page along with a statement whether or not the referenced resource is available. Operates on pages.

recall
: Displays a former version of a page (specified by its timestamp). This action is usually not called directly, but via `history`. Operates on pages.

referers
: Displays a list of pages referring to the specified page or file, operating on both object types.

deadlinks
: Displays a list of all pages that contain links to non-existing local resources. Operates on pages.

list
: Displays a list of local resources of various types.

search
: Evokes and interacts with Pie's search engine.

Authentication

login
: Authenticates as a registered user.

logout
: Logout from the site.

register
: Register a new user.

Editing Mode

The editing-mode is used by authors to enter and modify their contents after having logged in.

create
: Creates a new page.

add
: Actually adds the new page to the local archive. This action is usually not called by itself, but by `create`.

edit
: Edit an existing page.

update
: Store the changes made to a page in the local archive. This action is usually not called by itself, but by `edit`.

cancel
: Cancel editing or creating a page, removing the applied lock. This action is usually not called by itself, but by `edit` or `create`.

upload
: Upload a file.

overwrite
: Overwrite a local file. This action is usually not called by itself, but by `upload`.

delete
: Delete a local resource. Operates on pages and files.

remove
: Actually removes a local resource, or, in the case of pages, moves it to the trashcan. This action is usually not called by itself, but by `delete`.

rebuild
: Rebuild the output cache of a single page. Operates on pages.

rename
: Renames a page or file.

recycle

Takes a page out of the trashcan and re-uses it as a visable page. This action is usually not called by itself, but by `create`. Operates on pages only.

revert

Reactivate a previous version of a page. This action is usually not called by itself, but by `history`. Operates on pages only.

unlock

Release a lock created to edit or create a page. This action is usually not called by itself, but just serves as a means to manually unlock one's own pages after forgetting to properly `cancel` a previous operation.

Administrator Mode

The administrator mode is used by the administrator, that is, the superuser, to keep track of things and set things right (or wrong). The administrator is given handy and powerful tools that often affect the site as whole. As a rule of thumb and to make sure not to ruin aeons of work of other users, the underlying functionality should never be called manually, but only through the Maintenance Panel, even if this involves some pointing and clicking.

- Pitt Murmann, April 2006