PyGopherd Manual

PyGopherd Manual

Table of Contents

Chapter 1. Introduction to PyGopherd

Welcome to PyGopherd. In a nutshell, PyGopherd is a modern dynamic multi-protocol hierarchical information server with a pluggable modularized extension system, full flexible caching, virtual files and folders, and autodetection of file types -- all with support for standardized yet extensible per-document metadata. Whew! Read on for information on what all these buzzwords mean.

1.1. Features

Here are some of PyGopherd's features:

- Provides built-in support for multiple protocols: HTTP (Web), Gopher+, Gopher (RFC1436), Enhanced Gopher0, Gemini, Spartan, and WAP (mobile phones). Protocols can be enabled or disabled as desired.
- Provides protocol autodetection. That is, PyGopherd can listen for all the above protocols *on a single port* and will automatically respond using the protocol it detects the client is using. Practical effects of this are that you can, for instance, give out a single URL and have it viewable normally on desktop Web browsers and in WAP mode on mobile phones -- and appropriately in various Gopher browsers.
- Metadata and site links can be entered in a variety of formats, including full UMN dotfile metadata formats as well as Bucktooth gophermap files. Moreover, gophermap files are not limited to Gopher protocols, and can be used for all protocols.
- Support for inter-protocol linking (linking from Gopher sites to web sites)
- Virtual folder system lets you serve up anything as if it were regular files and directories. PyGopherd comes with the following virtual folder systems built in:
  - Can present any Unix MBOX, MMDF box, MH directory, Maildir directory, or Babyl mailbox as a virtual folder, the contents of which are the messages in the mailbox.
  - Can use a configurable separator to split a file into multiple parts, the first line of each becoming the name for the virtual folder.
  - Can peek inside a ZIP file and serve it up as first-class site citizens -- metadata can even be stored in the ZIP files.
  - Can serve up the contents of a dictd server as a filesystem.
- Modular, extensible design: you can use PyGopherd's own PYG extension format, or UMN- or Bucktooth-style executables.
- Runs on any platform supported by Python 2.2, 2.3, or 2.4. This includes virtually every past and current flavor of Unix (Linux, *BSD, Solaris, SunOS), Windows, MacOS 9.x and X, and more. Some features may not be available on non-Unix platforms.
- Runs on any platform supported by Java 1.1 via the Jython Python implementation.
- Tunable server types via configuration directive -- forking or threading.

- Secure design with support for chrooted execution.
- Feature-complete, full implementations of: Gopher0 (RFC1435), Gopher+, HTTP, and WAP.
- Support for automatically finding the titles of HTML documents for presentation in a directory.
- Versatile configuration file format is both extensible and nicely complementary of the module system.
- Protocol-independent, handler-dependent caching. This increases performance by letting handlers cache dynamically-generated information -- currently used by the directory handlers. This can improve performance of directories by several orders of magnitude. Because this is a handler cache only, all protocols share the single cache. Since the processing time for the protocols is negligible, this works out very well.
- Autosensing of MIME types and gopher0 item types. Both are completely configurable. MIME type detection is done using a standard mime.types file, and gopher0 types are calculated by using a configurable regexp-based MIME-to-gophertype map.
- Heavy support of regular expressions in configuration.
- ProtocolMultiplexer and HandlerMultiplexer let you choose only those protocols and handlers that you wish your server to support and the order in which they are tried when a request comes in.
- Full logging via syslog.

1.2. About Gopher

PyGopherd started life as a server for the Gopher Internet protocol. With Gopher, you can mount a filesystem (viewing files and folders as if they were local), browse Gopherspace with a web browser, download files, and be interactive with searching.

But this is only part of the story. The world of Gopher is more expansive than this. There are two major gopher protocols: Gopher0 (also known as RFC1436) and Gopher+. Gopher0 is a small, simple, lightweight protocol that is very functional yet also extremely easy to implement. Gopher0 clients can be easily placed in small embedded devices or in massive environments like a modern web browser.

Gopher+ is based on Gopher0 but extends it by providing document metadata such as file size and MIME type. Gopher+ allows all sorts of neat features, such as configurable metadata (serving up a bunch of photos? Add a Subject field to your metadata to let a customized photo browser display who is pictured) and multiple views of a file (let the user select to view your photos as PNG or JPEG).

Chapter 2. Quick Start

If you have already installed PyGopherd system-wide, or your administrator has done that for you, your task for setting up PyGopherd for the first time is quite simple. You just need to set up your configuration file, make your folder directory, and run it!

You can quickly set up your configuration file. The distribution includes two files of interest: `conf/pygopherd.conf` and `conf/mime.types`. Debian users will find the configuration file pre-installed in `/etc/pygopherd/pygopherd.conf` and the `mime.types` file provided by the system already.

Open up `pygopherd.conf` in your editor and adjust to suit. The file is heavily commented and you can refer to it for detailed information. Some settings to take a look at include: detach, pidfile, port, usechroot, setuid, setgid, and root. These may or may not work at their defaults for you. The remaining ones should be fine for a basic setup.

Invoke PyGopherd with **pygopherd path/to/configfile** (or **/etc/init.d/pygopherd start** on Debian). Place some files in the location specified by the root directive in the config file and you're ready to run!

Chapter 3. Installation

If you are reading this document via the "man" command, it is likely that you have no installation tasks to perform; your system administrator has already installed PyGopherd. If you need to install it yourself, you have three options: a system-wide installation with Debian, system-wide installation with other systems, and a single-user installation. You can download the latest version of PyGopherd from https://github.com/michael-lazar/pygopherd

3.1. Debian System-Wide Installation

You may install PyGopherd by simply running this command as root:

apt-get install pygopherd

3.2. Other System-Wide Installation

Download the tar.gz version of the package from the website. Make sure you have Python 3.7 or above installed; if not, download and install it from http://www.python.org/. Then run these commands:

```
tar -zxvf pygopherd-x.y.z.tar.gz
cd pygopherd-x.y.z
python3 setup.py
```

Next, proceed to configuration. Make sure that the `/etc/pygopherd/pygopherd.conf` file names valid users (setuid and setgid options) and a valid document root (root option).

You will type `pygopherd` to invoke the program.

3.3. Single-Account Installation

Download the tar.gz version of the package from the website. Make sure you have Python 2.2 or above installed; if now, download and install it from http://www.python.org/. Then run these commands:

```
tar -zxvf pygopherd-z.y.z.tar.gz
cd pygopherd-x.y.z
```

Modify `conf/pygopherd.conf` as follows:

- Set usechroot = no
- Set root to something appropriate.
- Set port to a number greater than 1024.

When you want to run PyGopherd, you will issue the **cd** command as above and then type **PYTHONPATH=. bin/pygopherd**. There is no installation step necessary.

Chapter 4. Configuration

PyGopherd is regulated by a configuration file normally stored in `/etc/pygopherd/pygopherd.conf`. You can specify an alternate configuration file on the command line. The PyGopherd distribution ships with a sample `pygopherd.conf` file that thoroughly documents the configuration file options and settings.

Chapter 5. Handlers

PyGopherd defines several handlers which are responsible for finding data on your server and presenting it to the user. The handlers are used to generate things like links to other documents and directory listings. They are also responsible for serving up regular files and even virtual folders.

Handlers are specified with the handlers option in `pygopherd.conf`. This option is a list of handlers to use. For each request that arrives, PyGopherd will ask each handler in turn whether or not it can handle the request, and will handle the request according to the first handler that is capable of doing so. If no handlers can handle the request, a file not found error is generated. See the example configuration file for an example.

The remaining parts of this section describe the different handlers that ship with PyGopherd.

5.1. dir.DirHandler

This handler is a basic one that generates menus based on the contents of a directory. It is used for directories that contain neither a `gophermap` file nor UMN-style links files, or situations where you have no need for either of those.

This handler simply reads the contents of your on-disk directory, determines the appropriate types of each file, and sends the result to the client. The descriptions of each item are usually set to the filename, but the html.HTMLFileTitleHandler may override that.

5.2. gophermap.BuckGophermapHandler

This handler is used to generate directory listings based on `gophermap` files. It will not read the directory on-disk, instead serving content from the `gophermap` file only. Gophermaps are useful if you want to present a directory in which the files do not frequently change and there is general information to present. Overall, if you only wish to present information particular to certain files, you should consider using the abstract feature of UMN.UMNDirHandler.

The `gophermap` files contain two types of lines, which are described here using the same convention normally used for command line arguments. In this section, the symbol \t will be used to indicate a tab character, Control-I.

`full line of informational text`

`gophertypeDESCRIPTION` [\t*`selector`* [\t*`host`* [\t*`port`*]]]

Note: spaces shown above are for clarity only and should not actually be present in your file.

The informational text must not contain any tab characters, but may contain spaces. Informational text will be rendered with gopher type i, which will cause it to be displayed on a client's screen at its particular position in the file.

The second type of line represents a link to a file or directory. It begins with a single-character Gopher type (see Gopher Item Types below) followed immediately by a description and a tab character. There is no space or other separator between the gopher type and the description. The description may contain spaces but not tabs.

The remaining arguments are optional, but only to the extent that arguments may be omitted only if all arguments after them are also omitted. These arguments are:

`selector`

: The selector is the name of the file on the server. If it begins with a slash, it is an absolute path; otherwise, it is interpreted relative to the directory that the gophermap file is in. If no selector is specified, the description is also used as the selector.

`host`

: The host specifies the host on which this resource is located. If not specified, defaults to the current server.

`port`

: The port specifies the port on which the resource is located. If not specified, defaults to the port the current server is listening on.

An example of a gophermap to help illustrate the concept is included with the PyGopherd distribution in the file `examples/gophermap`.

5.3. file.CompressedFileHandler

In order to save space, you might want to store documents on-disk in a compressed format. But then clients would ordinarily have to decompress the files themselves. It would be nice to have the server automatically decompress the files on the fly, sending that result to the client. That's where file.CompressedFileHandler comes in.

This handler will take compressed files, pipe them through your chosen decompression program, and send the result directly to clients -- completely transparently.

To use this handler, set the decompressors option in the configuration file. This option defines a mapping from MIME encodings (as defined with the encoding option) to decompression programs. Files that are not encoded, or which have an encoding that does not occur in the decompressors map, will not be decompressed by this handler.

Please see the sample configuration file for more examples and details about the configuration of this handler.

5.4. file.FileHandler

The file.FileHandler is just that -- its duty is to serve up regular files to clients.

5.5. html.HTMLFileTitleHandler

This handler is used when generating directories and will set the description of HTML files to the HTML title defined in them rather than let it be the default filename. Other than that, it has no effect. UMN gopherd implements a similar policy.

5.6. mbox handlers

There are four mailbox handlers:

- mbox.MaildirFolderHandler
- mbox.MaildirMessageHandler
- mbox.MBoxMessageHandler
- mbox.MBoxFolderHandler

These four handlers provide a unique "virtual folder" service. They allow you to present mailboxes as if they were folders, the items of the folders being the messages in the mailbox, organized by subject. This is useful for presenting mail archives or just making e-mail accessible in a nice and easy fashion.

To use these handlers, all you have to do is enable them in your handlers section. They will automatically detect requests for mailboxes and handle them appropriately.

The different handlers are for traditional Unix mbox mailboxes (all messages in a single file) and new qmail-stype Maildir mailboxes. You can enable only the two handlers for the specific mailbox type that you use, if desired.

5.7. pyg.PYGHandler

PYG (short for PYGopherd) is a mechanism that provides a tremendous amount of flexibility. Rather than just letting you execute a script like other Gopher or HTTP servers, PYGs are actually loaded up into PyGopherd and become fully-capable first-class virtual handlers. Yet they need not be known ahead of time, and are loaded dynamically.

With a PYG handler, you can generate gopher directories, handle searches, generate files, and more on the fly. You can create entire virtual directory trees (for instance, to interface with NNTP servers or with DICT servers), and access them all using the standard Gopher protocol. All of this without having to modify even one line of PyGopherd code.

If enabled, the pyg.PYGHandler will look for files with the extension .pyg that are marked executable. If found, they will be loaded and run as PYGs.

Please note: this module provides the capability to execute arbitrary code. Please consider the security ramifications of that before enabling it.

See the virtual.Virtual handler for more information about passing data to your scripts at runtime.

At present, documentation on writing PYGs is not provided, but you may find examples in the pygfarm directory included with the PyGopherd distribution.

5.8. scriptexec.ExecHandler

This handler implements "old-style" script execution; that is, executing arbitrary programs and piping the result to the client. It is, for the most part, compatible with both scripts written for UMN gopherd and the Bucktooth gopher server. If enabled, it will execute any file that is marked executable in the filesystem. It will normally list scripts as returning plain text, but you may create a custom link to the script that defines it as returning whatever kind of file you desire. Unlike PYGs, this type must be known in advance.

The scriptexec.ExecHandler will set environment variables for your scripts to use. They are as follows:

SERVER_NAME

: The name of this server as defined in the configuration file or detected from the operating system.

SERVER_PORT

: The port this server is listening on.

REMOTE_ADDR

: The IP address of the client.

REMOTE_PORT

The port number of the client.

REMOTE_HOST

The same value as REMOTE_ADDR

SELECTOR

The file that was requested; that is, the relative path to this script. If the selector included additional parameters after a |, they will be included in this string as well.

REQUEST

The "base" part of the selector; that is, the part leading up to the |.

SEARCHREQUEST

Included only if the client specified search data, this is used if the client is searching for something.

See the virtual.Virtual handler for more information about passing data to your scripts at runtime.

Please note: this module provides the capability to execute arbitrary code. Please consider the security ramifications of that before enabling it.

5.9. UMN.UMNDirHandler

This is one of the most powerful workhorse handlers in PyGopherd. It is designed to emulate most of the ways in which the UMN gopherd distribution generates directories, even going so far as to be bug-compatible in some cases. Generating directories with this handler is often the best general-purpose way to make nice directories in gopherspace.

The remainder of the description of the UMN.UMNDirHandler, except for the Abstracts and Info section, is lifted directly from the original UMN gopherd documentation, with light editing, because this handler implements it so exactly that there was no point in rewriting all that documentation :-)

5.9.1. Links

You can override the default view of a directory as generated by dir.DirHandler by creating what are known as *Links* in the data tree.

The ability to make links to other hosts is how gopher distributes itself among multiple hosts. There are two different ways to make a link. The first and simplest is to create a link file that contains the data needed by the server. By default all files in the gopher data directory starting with a period are taken to be link files. A link file can contain multiple links. To define a link you need to put five lines in a link file that define the needed characteristics for the document. Here is an example of a link.

```
Name=Cheese Ball Recipes
Numb=1
Type=1
Port=150
Path=1/Moo/Cheesy
Host=zippy.micro.umn.edu
```

The Name= line is what the user will see when cruising through the database. In this case the name is "Cheese Ball Recipes". The "Type=" defines what kind of document this object is. For a list of all defined types, see Gopher Item Types below. For Gopher+ and HTTP, a MIME type is also used, which is determined automatically based on the type you specify.

The "Path=" line contains the selector string that the client will use to retrieve the actual document. The Numb= specifies that this entry should be presented first in the directory list (instead of being alphabetized). The "Numb=" line is optional. If it is present it cannot be the last line of the link. The "Host=" and "Port=" lines specify a fully qualified domain name (FQDN) and a port respectively. You may substitute a plus '+' for these two parameters if you wish. The server will insert the current hostname and the current port when it sees a plus in either of these two fields.

An easy way to retrieve links is to use the Curses Gopher Client. By pressing '=' You can get information suitable for inclusion in a link file.

5.9.2. Overriding Defaults

The server looks for a directory called `.cap` when parsing a directory. The server then checks to see if the `.cap` directory contains a file with the same name as the file it's parsing. If this file exists then the server will open it for reading. The server parses this file just like a link file. However, instead of making a new object, the parameters inside the `.cap/` file are used to override any of the server supplied default values.

For instance, say you wanted to change the Title of a text file for gopher, but don't want to change the filename. You also don't want it alphabetized, instead you want it second in the directory listing. You could make a set-aside file in the `.cap` directory with the same filename that contained the following lines:

```
Name=New Long Cool Name
Numb=2
```

An alternative to `.cap` files are extended link files. They work just the same as the files described in Links above, but have a somewhat abbreviated format. As an example, if the name of the file was `file-to-change`, then you could create a file called `.names` with the following contents:

```
Path=./file-to-change
Name=New Long Cool Name
Numb=2
```

5.9.3. Adding Cool Links

One cool thing you can do with .Links is to add neato services to your gopher server. Adding a link like this:

```
Name=Cool ftp directory
Type=h
Path=/URL:ftp://hostname/path/
Host=+
Port=+

Name=Cool web site
Type=h
Path=/URL:http://hostname/
Host=+
Port=+
```

Will allow you to link in any FTP or Web site to your gopher. (See url.URLHandler for more details.)

You can easily add a finger site to your gopher server thusly:

```
Name=Finger information
Type=0
Path=lindner
Host=mudhoney.micro.umn.edu
Port=79
```

5.9.4. Hiding an Entry

This kind of trick may be necessary in some cases, and thus for object "fred", the overriding .names file entry would be:

```
Type=X
Path=./fred
```

by overriding default type to be "X". This may be useful, when for some reason there are symlinks (or whatever) in the directory at which PyGopherd looks, and those entries are not desired to be shown at all.

5.9.5. Abstracts and Info

Many modern gopher server maintainers like to intersperse gopher directory listings with other information -- often, additional information about the contents of files in the directory. The gophermap system provides one way to do that, and abstracts used with UMN gopher directories provides another.

Subject to the abstract_headers and abstract_entries configuration file options, this feature allows you to define that extra information. You can do that by simply creating a file named `filename.abstract` right alongside the regular file in your directory. The file will be interpreted as the abstract. For a directory, create a file named `.abstract` in the directory. Simple as that!

5.10. url.HTMLURLHandler

PyGopherd provides ways for you to link to pages outside Gopherspace -- that is, web pages, FTP sites, and the like. This is accomplished according to the Links to URL (http://lists.complete.org/gopher@complete.org/2002/02/msg00033.html.gz) specification (see Conforming To below for details). In order to link to a URL (EXCEPT gopher URLs) from a menu, you create a link of type h (regardless of the actual type of the resource that you are linking to) in your `gophermap` or `.Links` file that looks like this:

```
/URL:http://www.complete.org/
```

Modern Gopher clients that follow the Links to URL specification will automatically follow that link when you select it. The rest need some help, and that's where this handler comes in.

For Gopher clients that do not follow the Links to URL specification, the url.HTMLURLHandler will automatically generate an HTML document for them on the fly. This document includes a refresh code that will send them to the proper page. You should not disable this handler.

5.11. url.URLTypeRewriter

Some people wish to serve HTML documents from their Gopher server. One problem with that is that links in Gopherspace include an extra type character at the beginning, whereas links in HTTP do not. This handler will remove the extra type character from HTTP requests that come in, allowing a single relative-to-root link to work for both.

5.12. virtual.Virtual

This handler is not intended to ever be used directly, but is used by many other handlers such as the mbox support, PYG handlers, and others. It is used to generate virtual entries in the directory hierarchy -- that is, entries that look normal to a client, but do not actually correspond to a file on disk.

One special feature of the virtual.Virtual handler is that you can send information to it at runtime in a manner similar to a CGI script on the web. You do this by adding a question mark after the regular selector, followed by any arbitrary data that you wish to have sent to the virtual request handler.

5.13. ZIP.ZIPHandler

Using zip.ZIPHandler, you can save space on your server by converting part or all of your site into a ZIP file. PyGopherd can use the contents of that ZIP file as the contents of your site -- completely transparently.

The ZIP file handler must be enabled in the configuration file for this to work.

Chapter 6. Gopher Item Types

When you construct links to files via `.Links` or `gophermap` files, or modify the mapping in the configuration file, you will need to know these. Items bearing the "not implemented" text are not served up by PyGopherd as it ships, generally due to requirements of customized per-site software, but may be served up via PYG extension modules or other gopher servers.

This list was prepared based on RFC1436, the UMN gopherd(1) manpage, and best current practices.

0
: Plain text file

1
: Directory

2
: CSO phone book server (not implemented by PyGopherd)

3
: Error condition; text that follows is plain text

4
: Macintosh file, BinHex format

5
: DOS binary archive (not implemented by PyGopherd; use type 9 instead)

6
: uuencoded file; not directly generated by PyGopherd automatically, but can be linked to manually. Most gopher clients will handle this better as type 1.

7
: Search

8
: Telnet link

9
: Binary file

+
: Redundant server (not implemented by PyGopherd)

c

Calendar (not implemented by PyGopherd)

e

Event (not implemented by PyGopherd)

g

GIF-format graphic

h

HTML file

I

Any kind of graphic file other than GIF

i

Informational text included in a directory that is displayed but does not link to any actual file.

M

MIME multipart/mixed file

s

Any kind of sound file

T

tn3270 link

X
-

UMN-specific -- signifies that this entry should not be displayed in a directory entry, but may be accessed via a direct link. This value is never transmitted in any Gopher protocol.

I. PyGopherd Manpage

pygopherd

<jgoerzen@complete.org>

Name

PyGopherd — Multiprotocol Information Server

Synopsis

pygopherd [*configfile*]

Description

Welcome to PyGopherd. In a nutshell, PyGopherd is a modern dynamic multi-protocol hierarchical information server with a pluggable modularized extension system, full flexible caching, virtual files and folders, and autodetection of file types -- all with support for standardized yet extensible per-document metadata. Whew! Read on for information on this what all these buzzwords mean.

Quick Start

If you have already installed PyGopherd system-wide, or your administrator has done that for you, your task for setting up PyGopherd for the first time is quite simple. You just need to set up your configuration file, make your folder directory, and run it!

You can quickly set up your configuration file. The distribution includes two files of interest: `conf/pygopherd.conf` and `conf/mime.types`. Debian users will find the configuration file pre-installed in `/etc/pygopherd/pygopherd.conf` and the `mime.types` file provided by the system already.

Open up `pygopherd.conf` in your editor and adjust to suit. The file is heavily commented and you can refer to it for detailed information. Some settings to take a look at include: detach, pidfile, port, usechroot, setuid, setgid, and root. These may or may not work at their defaults for you. The remaining ones should be fine for a basic setup.

Invoke PyGopherd with **pygopherd path/to/configfile** (or **/etc/init.d/pygopherd start** on Debian). Place some files in the location specified by the root directive in the config file and you're ready to run!

Options

All PyGopherd configuratoin is done via the configuration file. Therefore, the program has only one command-line option:

`configfile`

This option argument specifies the location of the configuration file that PyGopherd is to use.

Conforming To

- The Internet Gopher Protocol as specified in RFC1436
- The Gopher+ upward-compatible enhancements to the Internet Gopher Protocol from the University of Minnesota as laid out at gopher://gopher.quux.org/0/Archives/mirrors/boombox.micro.umn.edu/pub/gopher/gopher_protocol/Gopher+/Gopher+.txt.
- The gophermap file format as originally implemented in the Bucktooth gopher server and described at gopher://gopher.floodgap.com/0/buck/dbrowse%3Ffaquse%201.
- The Links to URL specification as laid out by John Goerzen at gopher://gopher.quux.org/0/Archives/Mailing%20Lists/gopher/gopher.2002-02%3f/MBOX-MESSAGE/34.
- The UMN format for specifying object attributes and links with .cap, .Links, .abstract, and similar files as specified elsewhere in this document and implemented by UMN gopherd.
- The PYG format for extensible Python gopher objects as created for PyGopherd.
- Hypertext Transfer Protocol HTTP/1.0 as specified in RFC1945
- Hypertext Markup Language (HTML) 3.2 and 4.0 Transitional as specified in RFC1866 and RFC2854.
- Maildir as specified in http://www.qmail.org/qmail-manual-html/man5/maildir.html and http://cr.yp.to/proto/maildir.html.
- The mbox mail storage format as specified in http://www.qmail.org/qmail-manual-html/man5/mbox.html.
- Registered MIME media types as specified in RFC2048.
- Script execution conforming to both UMN standards as laid out in UMN gopherd(1) and Bucktooth standards as specified at gopher://gopher.floodgap.com:70/0/buck/dbrowse%3ffaquse%202, so far as each can be implemented consistent with secure design principles.
- Standard Python 2.2.1 or above as implemented on POSIX-compliant systems.
- WAP/WML as defined by the WAP Forum.

Bugs

Reports of bugs should be sent via e-mail to the PyGopherd issue tracker at https://github.com/michael-lazar/pygopherd/issues.

The Web site also lists all current bugs, where you can check their status or contribute to fixing them.

Copyright

Author

PyGopherd, its libraries, documentation, and all included files (except where noted) was written by John Goerzen <`jgoerzen@complete.org`> and copyright is held as stated in the Copyright section.

Portions of this manual (specifically relating to certian UMN gopherd features and characteristics that PyGopherd emulates) are modified versions of the original gopherd(1) manpage accompanying the UMN gopher distribution. That document is distributed under the same terms as this, and bears the following copyright notices:

PyGopherd may be downloaded, and information found, from its homepage:

https://github.com/michael-lazar/pygopherd

See Also

python (1).