W3 User's Manual

Second Edition, W3 Version 2.1

April 1994

William M. Perry
wmperry@indiana.edu

Copyright © 1993, 1994 William M. Perry

Permission is granted to make and distribute verbatim copies of this manual provided the copyright notice and this permission notice are preserved on all copies.

Introduction

W3 is an Emacs subsystem that allows the user to browse the wonderful World Wide Web.

The World Wide Web was started at the CERN physics institute in Switzerland in 1991. The project was initially started by Tim Berners-Lee (*timbl@info.cern.ch*) for distributing data between different research groups effectively.

The Web has since grown into the most advanced information system currently on the internet. It is now a global hypertext system with servers and browsers (programs written to interpret the hypertext language and display it correctly, and allow the user to follow links) exist for all major platforms (VMS, Windows, DOS, Unix, VM, NeXTstep, and Macintosh).

The basic concepts used in the Web are **hypertext** and **hypermedia**. Hypertext is the same as regular text, with one exception - it can contain links (cross-references) to other textual documents. Hypermedia is slightly different - it can contain links to other forms of media (movies, sounds, interactive programs, etc).

WWW also allows searches of indices that are located anywhere on the network; in this respect, it mirrors certain capabilities found in both WAIS and Gopher.

Client Side View of WWW

The WWW world consists of documents, and links. Indexes are special documents which, rather than being read, may be searched. The result of such a search is another *virtual* document containing links to the documents found. A simple protocol, *HTTP* is used to allow a browser program to request a keyword search by a remote information server.

The web contains documents in many formats. Those documents which are hypertext, (real or virtual) contain links to other documents, or places within documents. All documents, whether real, virtual or indexes, look similar to the reader and are contained within the same addressing scheme.

Information Provider View of WWW

The WWW browsers can access many existing data systems via existing protocols (FTP, NNTP) or via HTTP and a gateway. In this way, the critical mass of data is quickly exceeded, and the increasing use of the system by readers and information suppliers encourage each other.

Providing information is as simple as running the WWW server and pointing it at an existing directory structure. The server automatically generates the a hypertext view of your files to guide the user around.

To personalize it, you can write a few **SGML** hypertext files to give an even more friendly view. Also, any file available by anonymous FTP, or any internet newsgroup can be immediately linked into the web. The very small start-up effort is designed to allow small contributions. At the other end of the scale, large information providers may provide an HTTP server with full text or keyword indexing. This may allow access to a large existing database without changing the way that database is managed. Such gateways have already been made into Oracle(tm), WAIS, and Digital's VMS/Help systems, to name but a few.

The WWW model gets over the frustrating incompatibilities of data format between suppliers and reader by allowing negotiation of format between a smart browser and a smart server. This should provide a basis for extension into multimedia, and allow those who share application standards to make full use of them across the web.

Supported Markup Languages

Two different markup languages are supported by W3.

HTML

HTML is composed of a set of elements that define a document and guide its display. An HTML element may include a name, some attributes and some text or hypertext, and will appear in an HTML document as <tag_name> text </tag_name>,<tag_name attribute_name=argument> text </tag_name>, or just <tag_name>.

For example: '`<title> My Useful Document </title>`', and '`<pre width=60> A lot of text here. </pre>`'.

An HTML document is composed of a single element: <html>...</html>, that is, in turn, composed of head and body elements: <head>...</head>, and <body>...</body>. To allow older HTML documents to remain readable, <html>, <head>, and <body> are actually optional within HTML documents.

All the tags and attributes of HTML are fully supported in W3.

HTML+

The HTML+ language is an extension of HTML, with a large degree of backwards compatibility. See the internet draft by Dave Raggett for more information. The following HTML+ language elements are supported by W3:

- FORMS – forms are one of the most powerful aspects of HTML+. They allow the author of an HTML document to request information from the person viewing the document. Passwords, checkboxes, radio buttons, and many more types of entry fields are possible. W3 supports **all** the input tags, as well as GET and POST submission methods. W3 also supports the Mosaic 2.0 hack of automatically submitting forms with a NAME of isindex.
- ID tags for headers and paragraphs. This makes it possible to reference points within a document and not require empty hypertext links to do it.

- LINK support. LINKs allow you to specify the relevance between documents easily. This makes it possible to point to dictionaries, glossaries, or indices, or to specify the author of a document.
- Annotations. Can have <footnote> and <margin> notes that show up as footnotes at the bottom of a document.
- BR tag for explicit line breaks
- Notes and admonishments. A way to specify a warning or other standard information. Shows appropriate bitmap if possible. Similar to warning dialogs in Windows/Macintosh.
- Infinitely nested lists, and use of paragraphs and headers within lists.

Extra Markup

There are several different markup elements that are not officially part of HTML or HTML+ that W3 supports. These are either items that have been dropped from HTML+ because they would be too difficult for other browsers to implement or have been moved into another tag.

- RENDER support. The render tag can be used to convert one tag into another. Syntax is <RENDER TAG="PROPNAME" STYLE="I">, which will replace all <PROPNAME> tags with <I>, and all </PROPNAME> tags with </I>.
- Entity definitions within documents. This allows you to define per-document macros. Syntax is <!ENTITY TAG "Expansion of Tag">, which will replace all occurences of &TAG; with Expansion of Tag.
- Roman numerals for ordered lists. If an ordered list has an optional ROMAN attribute (`<OL ROMAN>`), then roman numerals will be used instead of normal arabic numbers.
- IMG and ALT support for list items. If a list item has an IMG or ALT tag (`<LI IMG="some.gif" ALT="text"`), then the list character will be replaced with either an icon or the ALT text.
- Specifying the hostname and port of an NNTP server in a news URL. URLs of the form news://hostname:port/messageID will work.

Supported Protocols

W3 supports the following protocols

Usenet News
: Can either display an entire newsgroup or specific articles by Message-ID: header. No parsing of your .newsrc is done, so beware of heavily trafficed newsgroups.

HTTP
: Supports both HTTP/0.9 and HTTP/1.0 protocols. Fully MIME-compliant with regards to HTTP/1.0. See [HTTP/1.0 Support], page 24

Gopher
: Support for all gopher types, include CSO searches.

Gopher+
: Support for Gopher+ retrievals. Support for converting ASK blocks into HTML+ FORMS and submitting them back to the server.

FTP
: FTP is handled by either ange-ftp or efs, the choice is up to the individual user.

Local files
: Local files are handled, and MIME content-types are derived from the file extensions.

Telnet
: Telnet is handled by running the emacs-lisp function `telnet`, or spawning an xterm running telnet.

TN3270
: TN3270 is handled by running a tn3270 program in an Emacs buffer, or spawning an xterm running tn3270.

Mail
: W3 supports the mailto: url scheme that causes a mail message to be started to a specific address.

Setting Up

This section of the manual will deal with getting, compiling, and configuring *Emacs-W3* for your site.

Retrieving W3

If you are using Lucid Emacs 19.9, you should be able to safely skip to the section on starting W3. W3 comes standard with all versions of Lucid Emacs from 19.9 onwards.

First off, you will need the file w3.tar.gz, which is available from cs.indiana.edu:/pub/elisp/w3 via anonymous ftp. If you do not have ange-ftp, html-mode, or the nntp package by Masanobu Umeda, then you will also need the extras.tar.gz.

After you have retrieved the files unpack them with the following commands: `zcat w3.tar.gz | tar xvvf -` and `zcat extras.tar.gz | tar xvvf -`. This will unpack distribution into two subdirectories '`w3`' and '`extras`'. To compile and install all the packages in the extras directory, please see the comments at the top of each lisp file.

Compiling W3

To install W3, go into the '`w3`' subdirectory and edit the '`Makefile`'. You need to change three variables:

`EMACS`
: This variable controls what version of Emacs is used to compile the programs. It should be the full path to the Emacs executable on your system. The default is to use Lucid Emacs ('`lemacs`').

`LISPDIR`
: This variable controls where the lisp code will be copied to when it is installed (with `make install`). This will usually be your personal lisp code directory (I prefer '`~/lisp`'). The value will be run through *expand-file-name* and then added to your load-path.

`DOTEMACS`
: This variable points to your Emacs customization file, usually '`~/.emacs`'. If you keep yours in a different place, you must change this variable. Otherwise `make emacs` will not work.

`INFODIR` This variable points to your local info directory (usually '`/usr/local/info`'). This can be any valid directory, as long as it is in `Info-default-directory-list` so that info-mode can find it.

`MAKEINFO` This variables controls how the info files are built. Possible values are `makeinfo` or `emacs -batch -q -f batch-texinfo-format`.

Once you have modified the '`Makefile`', you should be able to build several different targets.

`make w3` This will compile all the .el files into the much faster .elc files. If you are using Jamie Zawinski's optimizing byte compiler (standard in Emacs 19 and Lucid Emacs), then you may see a few compilation warnings (not many hopefully). These can be safely ignored as long as everything finishes compiling. This is the default target for `make` with no arguments.

`make install`
This will compile all the .el files and then copy .el and .elc files into the directory specified by `LISPDIR`.

`make emacs`
This will modify the file specified by `DOTEMACS`. A statement modifying the load-path variable and size autoload statements will be added to the end of the file.

`make all` This will compile and install the .el files, and also modify/create the `DOTEMACS` file.

`make w3.info`
This will create the emacs-readable info files. They will be created in the directory specified by `INFODIR`. The makefile variable `MAKEINFO` determines how the info file is built.

`make w3.dvi`
This will create the printable documentation, using tex and texindex to properly generate the indices. The w3.dvi file is left in the current directory.

Firewalls

Proxy Gateways

In late January, Kevin Altis and Lou Montulli proposed and implemented a new proxy service. This service requires the use of environment variables to specify a gateway server/port # to send protocol requests to. Each protocol (http, wais, gopher, ftp, etc) can have a different gateway server. The environment variables are *PROTOCOL*_proxy, where *PROTOCOL* is one of gopher, telnet, file, http, ftp, tn3270, news, mailto, www, or wais.

The main thing to understand about the proxy gateway is that instead of a partial URL being sent to the HTTP server, which is what we do today when a client talks directly to an HTTP server, a client must send a full URL (http://..., gopher://..., ftp://...) to the proxy gateway server, the rest of the HTTP message is the same. For gopher and ftp, the proxy gateway server will return the data encapsulated as a MIME content type to the client like a normal HTTP message. HTTP MIME content types are returned for all URL requests, regardless of the protocol type of the URL. FTP directories, Gopher directories, etc. are returned as text/html.

Basic Usage

W3 is similar to the Info package all emacs users hold near and dear to their heart (See section "Info" in *The Info Manual*, for a description of Info). Basically, `space` and `backspace` control scrolling, and `return` or `mouse2` will follow a hypertext link. The `f` and `b` keys maneuver around the various links on the page.

On non-graphic terminals (vt100, DOS, etc), or with a graphics terminal and old versions (18.xx) of emacs, hypertext links will be surrounded by '[[' and ']]' by default. On a graphics terminal with newer versions of emacs (epoch, lucid, or FSF 19), the links will be in bold print. See [Controlling Formatting], page 19 for information on how to change this, or for help on getting the highlighting to work on graphics terminals.

There are approximately 50 keys bound to special W3 functions. The basic rule of thumb regarding keybindings in W3 is that a lowercase key will take an action on the **current document**, and an uppercase key will take an action on the document pointed to by the hypertext link **under the cursor**.

There are several areas that the keybindings fall into: movement, information, action, and miscellaneous.

Movement

`SPC` Scroll downward in the buffer. With prefix arg, scroll down that many screenfuls.

`DEL` Scroll upward in the buffer. With prefix arg, scroll up that many screenfuls.

`<, M-x w3-start-of-document`
: Go to start of document

`>, M-x w3-end-of-document`
: Go to end of document

`b, M-x w3-back-link`
: Attempts to move backward one link area in the current document. Signals an error if no previous links are found.

`M-x w3-show-hotlist`
: Possibly go to a link in the hotlist. A hypertext listing of the items in the hotlist is generated on the fly, and the links can be followed normally.

`H, M-x w3-use-hotlist`
: Possibly go to a link in the hotlist. A new buffer is created for the new document. If the variable `w3-keep-old-buffers` is non-`nil`, the current buffer is kept. If `nil`, then the current buffer is killed.

`m, M-x w3-complete-link`
: Choose a link from the current buffer and follow it. A completing-read is done on all the links, so `space` and `TAB` can be used for completion.

`f, n, M-x w3-forward-link`
: Attempts to move forward one link area in the current document. Signals an error if no more links are found.

Information

These functions relate information about one or more links on the current document.

`v, M-x w3-view-url`
: This will show the URL of the current document in the minibuffer.

`V, M-x w3-view-this-url`
: This will show the URL of the hypertext link under point in the minibuffer. If there is not a hypertext link under point, then it will show the type of form entry area under point. If there is no form entry area under point, thn it will show the inlined image's URL that is under point, if any.

`s, M-x w3-source-document`
: This will show the HTML source of the current document in a separate buffer. The buffer-name will be based on the document's URL.

`S, M-x w3-source-document-at-point`
: This will show the HTML source of the hypertext link under point in a separate buffer. The buffer-name will be based on the document's URL.

`k, C-k, M-x w3-save-url`
: This will store the current document's URL in the kill ring, and also in the current window-system's clipboard, if possible.

`K`, `M-x w3-save-this-url`
: This will store the URL of the document under point in the kill ring, and also in the current window-system's clipboard, if possible.

Action

First, here are the keys and functions that will bring up a new hypertext page, usually creating a new buffer.

`return`
: Pressing return attempts to follow the link under the cursor.

`button2`, `M-x w3-follow-mouse`
: This function expects to be bound to a mouse button. It movess to the point under mouse and try to fetch the link that was clicked on. If no link is found, a message is displayed in the minibuffer.

`C-button2`, `M-return`, `M-x w3-follow-inlined-image`
: This function tries to retrieve the inlined image that is under point. It ignores any form entry areas or hyperlinks, and blindly follows any inlined image. Useful for seeing images that are meant to be used as hyperlinks when you are not on a terminal capable of displaying graphics.

`m`, `M-x w3-complete-link`
: This will do a completing-read on all the hypertext links in the current buffer. You can use `space` and `tab` to complete on the links.

`r`, `g`, `M-x w3-reload-document`
: This reloads the current document – the current buffer is killed, and the URL it was visiting is fetched and redisplayed. Your position within the buffer remains the same (unless the document has changed since you last retrieved it, in which case it should be relatively close).

`C-o`, `M-x w3-fetch`
: This function prompts for a URL in the minibuffer, and attempts to fetch it. If there are any errors, or W3 cannot understand the type of link you requested, the errors are displayed in a hypertext buffer.

`o`, `M-x w3-open-local`
: Find a local file, interactively. This prompts you for a local file name to open. The file must exist, and may be a directory. If the requested file is a directory and `w3-directory-format` is `'dired`, then a dired-mode buffer is displayed. If `'hypertext`, then W3 automatically generates a hypertext listing of the directory. `'hypertext` is the default, so that all the keys and functions remain the same.

`M-s, M-x w3-search`
: Perform a search, if this is a searchable index. This sends a string of the type ''`URL?search-terms`'' to the server this document was retrieved from. Searching requires a server, W3 can not do local file searching, as there are too many possible types of searches people could want to do.

`C-c C-b, M-x w3-show-history-list`
: If `w3-keep-history` is non-`nil`, then W3 keeps track of all the URLs you visit in an Emacs session. This function takes all the links that are in that internal list, and formats them as hypertext links in a list.

And here are the commands to move around between W3 buffers:

`l, M-x w3-goto-last-buffer`
: Go to last WWW buffer visited

`q, M-x w3-quit`
: Quits WWW mode. This kills the current buffer and goes to the most recently visited buffer.

`u, M-x w3-leave-buffer`
: This is similar to w3-quit, but the buffer is not killed, it is moved to the bottom of the buffer list (so it is the least likely to show up as the default with switch-to-buffer).

`B, M-x w3-backward-in-history`
: This will take you one step back along the path in the current history. Will have no effect if you are at the beginning of your history list.

`F, M-x w3-forward-in-history`
: This will take you one step forward along the path in the current history. Will have no effect if you are at the end of your history list.

Miscellaneous

`M-m, M-x w3-mail-current-document`
: This will mail the current document to someone. You can choose from 3 different formats to mail: formatted text, HTML source, or LaTeX source. If you mail the HTML source, then an appropriate `<base>` tag is inserted at the beginning of the document so that relative links may be followed correctly by whoever receives the mail.

`M-M, M-x w3-mail-document-under-point`

This will mail document pointed to by the hypertext link under point to someone. You can choose from 3 different formats to mail: formatted text, HTML source, or LaTeX source. If you mail the HTML source, then an appropriate <base> tag is inserted at the beginning of the document so that relative links may be followed correctly by whoever receives the mail.

`p, M-x w3-print-this-url`

This will print the current document. You can choose from 3 different formats to mail: formatted text, HTML source, or postscript (by using LaTeX and dvips).

If you print the formatted text, the normal `lpr-buffer` function is called, and the variables `lpr-command` and `lpr-switches` will control how the document is printed.

If you print the HTML source, then an appropriate <base> tag is inserted at the beginning of the document. If postscript is chosen, then the HTML source of the document is converted into LaTeX source. If the variable `w3-use-html2latex` is non-`nil`, then the program specified by `w3-html2latex-prog` is run in a subprocess with the arguments in `w3-html2latex-args`. The `w3-html2latex-prog` should accept HTML source on its standard input and send the LaTeX output to standard output. If `w3-use-html2latex` is `nil`, then an emacs lisp function will use regular expressions to replace the HTML code with LaTeX markup. The variable `w3-latex-docstyle` controls how the document will be laid out in this case, and postscript figures will be printed as well.

`P, M-x w3-print-url-under-point`

This will print the document pointed to by the hypertext link under point. Please see the documentation for `w3-print-this-url` directly above for more information.

`M-x w3-insert-formatted-url`

This will insert a fully formatted HTML link into another buffer. This will get the name and URL of either the current buffer, or, with a prefix arg, of the link under point, and construct the appropriate <a...>... markup and insert it into the desired buffer.

`M-tab, M-x w3-insert-this-url`

This will insert the URL of the current document into another buffer. Buffer is prompted for in the minibuffer. With prefix arg, uses the URL of the link under point.

`U, M-x w3-use-links`

Select one of the <LINK> tags from this document and fetch it. Links are attributes of a specific document, and can tell such things as who made the document, where a table of contents is located, etc.

This will allow you to follow the <link...> tags in the current document. Link tags specify relationships between documents in two ways. Normal (forward) relationships (where the link has a REL="xxx" attribute), and reverse relationships (where the link has a REV="xxx" attribute). This will first ask what type of link you want to follow (Normal or Reverse), then will do a completing-read on only the links that have that type of relationship.

Since NCSA Mosaic is the de-facto hypertext browser at most sites, W3 is compatible with it in several ways.

Hotlist Handling

In order to avoid having to traverse many documents to get to the same document over and over, W3 supports a "hotlist" like Mosaic. This is a file that contains URLs and aliases. You can quickly go to any document in the Web, providing you've been there before and added it to your hotlist. The variable `w3-hotlist-file` determines where this information is saved. The structure of the file is compatible with Mosaic's hotlist file, so this defaults to '`~/.mosaic-hotlist-default`'.

Hotlist commands are:

`a, M-x w3-hotlist-add-document`
: This adds the current document to your hotlist, with the buffer name as its identifier. Modifies the file specified by `w3-hotlist-file`.

`d, M-x w3-hotlist-delete`
: You will be prompted for the alias of the entry to kill. You may press the spacebar or tab key for partial completions. Once you select the item to delete, the internal representation of the hotlist is updated, and the file specified by `w3-hotlist-file` is updated.

`M-x w3-hotlist-rename-entry`
: Some hotlist item names can be very unwieldy ('Mosaic for X level 2 fill out form support'), or uninformative ('Index of /'). If you are not satisfied with how a specific item is labeled, you may change it by typing `M-x w3-rename-hotlist-entry`. You will be prompted for the item to rename in the minibuffer – you may use the spacebar or tab key for completion. After you have chosen an item to rename, you will be prompted for a new title until you enter one that does not already exists in the hotlist. This function modifies the file specified by `w3-hotlist-file`.

`H, M-x w3-use-hotlist`
: Prompts you for the alias to jump to. You may press the spacebar or tab key for partial completions.

`M-x w3-show-hotlist`
: This converts your hotlist into an ordered list and parse it as HTML.

History

NCSA Mosaic keeps track of the URLs you have followed from a page, so that it can provide **forward** and **back** buttons to keep a *path* of URLs that you can review easily. If you set the variable `w3-keep-history` to `t`, then W3 will keeep a list of all the URLs you visit in a session. To see a listing of your history for this session of W3, you can type `M-x w3-show-history` from any buffer, and W3 will generate an HTML document showing every URL you have visited since you started Emacs (or cleared the history list), and then format it. Any of the links can be chosen and followed to the original document. To clear the history list, you can choose 'Clear History' from the 'Options' menu.

Another twist on the history list mechanism is the fact that all W3 buffers remember what URL, buffer, and buffer position you were at before jumping to this document, and also keeps track of where you jump **to** from that buffer. This means that you can go forwards and backwards very easily along the path you took to reach a particular document. To go forward, use the function `w3-forward-in-history`, to go backward, use the function `w3-backward-in-history`. These are fairly stable functions, but may not work as expected all the time. First, the buffer-list is used to look at the URL of every buffer, and if it matches the item in the history list you are looking for, then it is brought forward. If no buffer containing the desired URL is found, then the URL is fetched. Then the desired position in the buffer is searched for.

Global History

Mosaic supports the idea of a "history" of URLs you have visited, and it displays them in a different style than normal URLs so you will not visit someplace again on accident.
If you set the variable `w3-keep-history` to `t`, then W3 will keeep a list of all the URLs you visit in a session. To write the file out, you can press `C-q` in a W3 buffer, or `M-x w3-write-global-history` in any buffer. The list will be added to those already in the file specified by `w3-global-history-file`, which defaults to '`~/.mosaic-global-history`'.

You no longer have to explicitly save the history file – w3 uses the `kill-emacs-hook` to write it out whenever you exit emacs.

If any URL in the list is found in the file, it is not saved, but new ones are added at the end of the file.

One of the nice things about keeping a global history files is that W3 can use it as a completion table. When doing `M-x w3-fetch`, you can hit the `tab` or `space` keys to complete on a partial URL. This is very useful, especially for very long URLs that you didn't add to your hotlist, or for seeing all the pages from a particular web server before choosing which you want to see.

Annotations

Mosaic allows you to *annotate* documents. You can type in comments about the current document, and these annotations will appear as a link to your comments at the end of the document when you browse it in Mosaic. The original file is not changed.There are two types of annotations supported in Mosaic, and both are supported by W3 as well.

Group Annotations

NOTE: The group annotation experiment has been terminated. NCSA is running the annotation server temporarily, but it will not be around much longer. It will be replaced with support on the server side for adding <LINK> tags to documents.

An experimental *annotation server* is running at NCSA. This server allows people to annotate any document on the Web, and have anyone else see it. To activate this feature in Mosaic, set the resource Mosaic*annotationServer to hoohoo.ncsa.uiuc.edu:8001.

In W3, if the variable `w3-group-annotation-server` is defined, then it will try to contact the server and retrieve any annotations for the current document. This variable should be a string containing the name of the annotation server (defaults to "hoohoo.ncsa.uiuc.edu"). To turn off group annotation support, set this variable to `nil`. If you have a slow connection to NCSA, I'd recommend doing this. The variable `w3-group-annotation-port` defaults to 8001, and should be an integer representing the port number the server is running on.

The group annotations (if any) will appear as normal links at the bottom of your document. They can be selected just like any other link.

Both Mosaic and W3 support multiple levels of annotation (Annotations on annotations ad infitim).

To create a new annotation, type `M-x w3-add-group-annotation`. This will put you in a new buffer, in HTML-mode. If you are not familiar with HTML-mode, hit `C-h m` for instructions, and see the file html-mode.el that came with W3. At the top of this buffer is an HTML tag (</PRE>). If you don't want to use any HTML commands, you can delete this and just type in your message and it should be formatted just as you type it. If you leave this in, you can use any HTML formatting command (italics, headings, bold, etc) except adding a title. When you are finished editing your annotation, type `C-c C-c`. W3 will then prompt you for a title for your annotation, defaulting to "Annotation by <user information>". Hit return after typing in your title. If any errors are received, they will be reported in a separate buffer.

To delete an annotation, you must be reading it. Then type `M-x w3-delete-group-annotation`. If any errors are received, they will be displayed in a separate buffer.

I do not currently support editing group annotations.

Personal Annotations

If you do not want to share your musings about a particular document with the entire network, you can add a personal annotation that only you can see. Personal annotations are stored in a subdirectory in the users account on the local disk, with a log file that contains information about what URLs have been annotated and which files contain the annotations.

W3 will look in the directory specified by `w3-personal-annotation-directory` (defaults to '`~/.mosaic-personal-annotations`'). Any personal annotations for a document will be automatically appended when it is retrieved.

To add a new personal annotation, type `M-x w3-add-personal-annotation`. This will put you in a new buffer, in HTML-mode. If you are not familiar with HTML-mode, hit C-h m for instructions, and see the file html-mode.el that came with W3. At the top of this buffer is an HTML tag </PRE>. If you don't want to use any HTML commands, you can delete this and just type in your message and it should be formatted just as you type it. If you leave this in, you can use any HTML formatting command (italics, headings, bold, etc) except adding a title. When you are finished editing your annotation, hit C-c C-c. It will then prompt you for a title for your annotation, defaulting to Annotation by <user information> on <date>.

To delete a personal annotation, you must be reading it. Once reading the annotation, type `M-x w3-delete-personal-annotation`. It will delete the file containing the annotation, and any references to it in your log file.

I do not currently support editing personal annotations.

Controlling Formatting

How W3 formats a document is very customizable. How a document is displayed depends on several things – whether the user is on a terminal capable of graphics, and a few variables.

The following sections describe in more detail how to change the formatting of a document.

General formatting conventions

Setting the fill column

Each time a document is parsed, the `fill-column` is recalculated using `window-width` and `w3-right-border`. `w3-right-border` is an integer specifying how much room at the right edge of the screen to leave blank. The `fill-column` is set to `(- (window-width) w3-right-border)`.

Formatting of hypertext links

If the variable `w3-delimit-links` is `non-nil` (the default for text-terminals), then hypertext links will be surrounded by text specified by the user. The variables `w3-link-start-delimiter` and `w3-link-end-delimiter` control what text is at the start and end of a hypertext link. These variables are cons-pairs of two strings.

If a link has never been visited before (it is not in the *global history*), then the `car` of these variables will be inserted at the start and end of the link. If the link has been visited before, then the `cdr` will be inserted. So, links will look like:

```
[[This is a hypertext link]] that has never been visited.
{{This one, however}} has been seen before at some point in time.
```

All reserved characters should be replaced with their HTML[+] entity definitions.

Formatting of lists

There are several different ways to control the formatting of lists. The most obvious is how deeply they are indented relative to the rest of the paragraphs in the document. To control this, you should set the variables `tab-stop-list` and `tab-width`. A tab every eight spaces is considered normal, but I prefer to have a tab every four spaces. So `tab-stop-list` would be ‘`'(4 8 12 16 20 24 28 32 36 ...)`’, and `tab-width` would be ‘4’. I use `setq-default` in my .emacs for these, but you can also use them from `w3-mode-hooks`. Something like:

```
(add-hook 'w3-file-prepare-hooks
          (function
            (lambda ()
              (setq tab-stop-width (list 4 8 16 20 24 28 32 36 40 44)
                    tab-width 4))))
```

Another thing that is easy to change about lists is the bullet character put at the front of each list item. This is controlled by the variable `w3-list-chars-assoc`, which is an assoc list. This is a list of lists, each sublist describing what to put at the start of each particular list type. The `car` of this list should be the type of list, in all caps (ie: ‘`UL`’). The rest of the list should consist of strings to insert at certain levels of lists. The `nth` element of this list will be used when the list is nested `n + 1` levels. If the list is not long enough to define a string for a certain nesting level, then it defaults to either a '*' or a '.'.

Formatting of directory listings

When W3 encounters a link to a directory (whether by local file access or via ftp), it can either create an HTML document on the fly, or use `dired-mode` to peruse the listing. The variable `w3-directory-format` controls this behavior.

If the value is `'hypertext`, use `directory-files` to list them out and transform them into a hypertext document, then pass it through the parser like any other document.

If the value is `'dired`, just pass the directory off to dired using find-file. Using this option loses all the hypertext abilities of W3, and you will be unable to load documents in the directory directly into W3 by clicking with the mouse, etc.

Formatting of gopher directories

There are two different ways of viewing gopher links. You can use the built-in support that converts gopher directories into HTML, or you can use the 'gopher.el' package by Scott Snyder <snyder@fnald0.fnal.gov>. The variable that controls this is `w3-use-hypertext-gopher`. If set to `nil`, then 'gopher.el' is used. Any other value causes W3 to use its internal gopher support. If you use 'gopher.el', you lose all the hypertext capabilities of W3. All the functionality of 'gopher.el' is now available in the hypertext version, and the hypertext version supports Gopher+ and ASK blocks.

The main way to control the display of gopher directories is by the variable `w3-gopher-labels`. This variable controls the text that is inserted at the front of each item. This is an assoc list of gopher types (as one character strings), and a string to insert just after the list item. All the normal gopher types are defined. Entries should be similar to: `'("0" . "(TXT)")'`. I have tried to keep all the tags to three characters plus two parentheses.

Creating a horizontal rule

Horizontal rules (**<HR>** tags in HTML[+]) are used to separate chunks of a document, and is meant to be rendered as a solid line across the page. Some terminals display characters differently, so the variable `w3-horizontal-rule-char` controls which character is used to draw a horizontal bar. This variable must be the ASCII value of the character, **not a string**. The variable is passed through make-string whenever a horizontal rule of a certain width is necessary.

On character based terminals

On character based terminals, there is no easy way to show that a certain range of text is in bold or italics. If the variable `w3-delimit-emphasis` is non-nil, then W3 can insert characters before and after character formatting commands in HTML documents. The defaul value of `w3-delimit-emphasis` is automatically set based on the type of window system and version of Emacs being used.

Two variables control what text is inserted around different markup tags. `w3-header-chars-assoc` controls what characters are inserted around header items, and `w3-style-chars-assoc` controls what characters are inserted around most other markup (italics, addresses, etc).

w3-header-chars-assoc is an assoc list of header tags and a list of formatting instructions. The car of the list is the level of the header (1 – 6). The rest of the list should contain three items. The first item is text to insert before the header. The second item is text to insert after the header. Both should have reserved characters converted to their HTML[+] entity definitions. The third item is a function to call on the area the header is in. This function is called with arguments specifying the start and ending character positions of the header. The starting point will always be first. To convert a region to upper case, please use w3-upcase-region instead of upcase-region, so that URLs within the region will not be corrupted.

w3-style-chars-assoc is an assoc list of style tags and a list of strings. The car of the list is the type of style tag it specifies (DFN, B, I, etc). The rest of the list should contain two items. The car is text to insert before the stylized text. The cdr is text to insert after the stylized text. Both should have reserved characters converted to their HTML[+] entity definitions.

With graphics workstations

When you are running in a graphic environemnt (Xwindows or NeXTstep for example), the fonts and colors used by W3 to display text can be controlled by setting a few resources. To specify these resources:

- Xwindows: Place them into your '~/.Xresources' file or the '/usr/lib/X11/app-defaults/Emacs' file.
- NeXTstep: If you are using the NeXTstep port of Emacs 19, you should use the 'dwrite' command to install these resources. The command should look something like: ('dwrite Emacs <Style>.Attribute<item> value', where style and attribute are one of the choices below.
- OS/2 or Windows: If you are using the Lucid Emacs or FSF Emacs that has been ported to the Presentation Manager or Windows 3.x, it will tell you where to store a file that Emacs will look at for Xresources.

The resources for each version of Emacs are the same. For each style of text that W3 uses, you can specify any of the following resources by replacing <style> with the actual style name.

- Emacs*<Style>.AttributeFont: Font name
- Emacs*<Style>.AttributeForeground: Foreground color
- Emacs*<Style>.AttributeBackground: Background color
- Emacs*<Style>.AttributeUnderline: Underline text?

W3 uses these style names:

`w3-address-style`
: For anything in <ADDRESS> tags.

`w3-bold-style`
: For anything in bold, strong, or a definition

`w3-default-style`
: For normal text

`w3-header-style`
: For any headers

`w3-italic-style`
: For italic text, emphasized, or addresses

`w3-node-style`
: For links to other documents

`w3-strikethru-style`
: For items that should be shown as struck-out.

`w3-subscript-style`
: For subscripted items

`w3-superscript-style`
: For superscripted items

`w3-tt-style`
: For fixed width fonts, code segments, samples, or variables

`w3-underline-style`
: For underlining, and citations

`w3-visited-node-style`
: For displaying hypertext links that have been viewed before

To determine what styles are displayed in what face, W3 uses an assoc list called `w3-style-assoc`. It is of the form '`(TAGNAME . FACENAME)`', where tagname is the HTML[+] tag that surrounds the text (without the <, >, or /), and FACENAME is either a symbol representing a face (in Emacs 19 and Lucid Emacs) or a symbol representing a variable that holds the face information (Epoch 4.x).

HTTP/1.0 Support

The new revision of the HTTP specification adds much more functionality to the server side of a transaction. Access authorization (via username and password combinations) has been added, and several types of redirection can occur. All of this negotiation and redirection should take place **before** the user ever sees the first requested page - this will avoid the overhead of parsing any error messages or old documents the server may have returned with the redirection or authorization message. The new protocol is also MIME (Multimedia Internet Mail Extensions, see RFC 1341) compliant.

Redirection

One of the most useful aspects of HTTP/1.0 is the ability to transparently move files between different servers (perhaps even different protocols). Most of the WWW browsers support redirection in some form or another. The Emacs browser supports all three types of redirection in the HTTP/1.0 specification (error codes 301, 302, and 303).

Whenever a redirection response is detected, the URL specified by the **Location:** header is retrieved. All relative references are resolved before requesting the new URL.

Eventually, an HTML editor that is tightly integrated with the browser is planned, and it will be able to edit documents to change their links if a permanent relocation is seen.

Authentication

Lots of information will be useful to a group of people within an organization, or a group working on a project, but it is not always wise to distribute this information to the world at large.

The HTTP/1.0 protocol adds the capability to have authentication based on usernames and passwords. If the improper username/password pair is sent to the server, an error code of 401, **Unauthorized** will be returned by the server.

The browser has a very extensible interface to its authentication handling. When a 401 error code is received, the *Auth-type* header is checked – this header field should be a space-separated list of suitable authorization schemes for the requested URL. The value of this header is read into a lisp symbol by way of Emacs' read-string function. This lisp symbol will be similar to `w3-`*authtype*`-auth`, where *authtype* is replaced by the correct authorization type. The authorization types defined in the latest HTTP/1.0 specification include user, basic, public key, and kerberos (versions 4 and 5). If a function of this name is currently defined, then the function is called via `funcall` with several parameters:

1. The URL being authenticated.
2. whether to prompt for a username/password if no cached data is found.
3. whether to overwrite an old username/password if the information is found in the cache.

This interface was chosen for its flexibility and extensibility. The main routine that does the MIME parsing and the buil,ding of the Authorization header does not need to know how to handle each type of authentication, and the addition of a new method for authentication is simply a matter of defining one function that conforms to a simple interface.

The only authorization supported by the browser at this time is 'basic.' This is simply a string that looks like '`user:password`' that has been encoded using Base64 encoding as defined in RFC 1421. It is given as an example of how to write an authorization module. All of the functions for storing, retrieving, and over-writing the cached authorization information should all be handled by one function (although it would be perfectly acceptable to have a stub function that passed off to three larger functions based on its parameters). The most efficient way to store the cached information is by an assoc-list of assoc-lists. The top level assoc list is keyed on the name of the server. The secondary assoc-list is keyed on the full path of the file that is protected. Thus, a sample authorization cache would look like this:

```
((''info.cern.ch''    . ((''/foo''          . ''d21wZXJyeTpOZXN0aW5n'')
                         (''/bar''          . ''amtvbnJhdGg6ZGlzbWVtYmVy'')
                         (''/foo/x.html'' . ''dmlvbGV0dDpvcGVuZ2w='')))
 (''cs.indiana.edu''  . ((''/elisp/w3/''  . ''dGxvb3M6Y29ucXVlcg=='')
                         (''/''           . ''bXZhbmhleW46a2lsbGh1bGljaw=='')))
)
```

The structure consists of two assoc-lists for the sake of speed. The list of cached information could conceivably hold several thousand links (if the user does not exit Emacs for long periods of time.) If the list were keyed on a full URL, the assoc function would have to search through every link before failing to find a new URL. With the current scheme, assoc only has to search though a few items (maximum is the number of HTTP servers, which will always be much, much smaller than the number of distinct URLs.) Even with a 3:1 ratio of URLs to each server, this will be a big win.

Payment

The Chargeto: header will be used to pay for services offered over the World Wide Web. Such things as electronic magazines and commercial databases will all need a way to restrict access to only authorized subscribers. The format of the header has yet to be specified, but the interface and storage techniques will be similar to the Authorization section.

MIME Support

MIME is an emerging standard for multimedia mail. It offers a very flexible typing mechanism. The type of a file/message is specified in two parts, separated by a '/'. The first part is a general category of data (text, application, image, etc). The second part is the specific type of data (postscript, gif, jpeg, etc). So '`text/html`' specifies an HTML document, whereas '`image/x-xwindowdump`' specifies an image of an Xwindow taken with the xwd program.

This typing allows much more flexibility in naming files. HTTP/1.0 servers can now send back content-type headers in response to a request, and not have the client second-guess it based on file extensions. Now you can name HTML files '`something.gif`' if you really felt like it (not a great idea, but doable).

Adding MIME types based on file extensions

For some protocols however, it is still necessary to guess the content of a file based on the file extension. This type of guess-work should only be needed when accessing files via FTP, local file access, or old HTTP/0.9 servers.

Instead of specifying how to view things twice, once based on content-type and once based on the file extension, it is easier to map file extensions to MIME content-types. The variable that controls this is `w3-mime-extensions`.

This variable is an assoc list of file extensions and the corresponding MIME content-type. A sample entry looks like: '`(".movie" . "video/x-sgi-movie")`' This makes all files that end in '`.movie`' ('`foo.movie`' and '`bar.movie`') be interpreted as SGI animation files. If a content-type is defined for the document, then this is over-ridden. Regular expressions can **NOT** be used.

Both Mosaic and the NCSA HTTP daemon rely on a separate file for mapping file extensions to MIME types. Instead of having the users of W3 duplicate this in lisp, this file can be parsed using the `w3-parse-mime-types` function. This function is called each time w3 is loaded, and tries to parse the file '`~/.mime-types`' if it exists. This format of this file is as follows.

Each line contains information for one http type. These types resemble MIME types. If you plan to add new ones, you should use subtypes beginning with x-, such as application/x-myprogram. Lines beginning with # are comment lines, and suitably ignored. Each line consists of:

type/subtype ext1 ext2 ... ext*n*

type/subtype is the MIME-like type of the document. ext* is any number of space-separated filename extensions which correspond to the MIME type.

Mapping gopher types to MIME types

Need to fill this in ::WORK::

Specifying Viewers

Not all files look as they should when parsed as an HTML document (whitespace is stripped, paragraphs are reformatted, and lots of little changes that make the document look unrecognizable). Files may be passed to external programs or Emacs lisp functions to be viewed.

Not all files can be viewed accurately from within an Emacs session (GIF files for example, or audio files). For this reason, the user can specify file "viewers" based on MIME content-types. The variable `w3-mime-viewers` is an assoc-list contains MIME content-types and a lisp object specifying how to view a document.

If the lisp object is a string (ie: '`"xv -perfect %s"`'), it specifies an external program that should be used to view the document. This is passed through the `format` function. The first %s will be replaced with the temporary file name the retrieved document is saved in, and then that resulting string is used as a command in a subprocess via `start-process`.

If the lisp object is a lisp symbol (ie: '`tar-mode`'), it species a lisp function to call after visiting the buffer. This is appropriate for using major modes to view documents. The function is called with no arguments.

If the lisp object is a list (ie: '`(message "wow")`'), it specifies something to be passed through `eval`.

Mailcap File

Both Mosaic and the NCSA HTTP daemon rely on a separate file for mapping file extensions to MIME types. Instead of having the users of W3 duplicate this in lisp, this file can be parsed using the `w3-parse-mime-types` function. This function is called each time w3 is loaded, and tries to parse the file '`~/.mime-types`' if it exists. This format of this file is specified in RFC 1343, but a brief synopsis follows (this is taken verbatim from sections of RFC 1343).

Each mailcap file consists of a set of entries that describe the proper handling of one media type at the local site. For example, one line might tell how to display a message in Group III fax format. A mailcap file consists of a sequence of such individual entries, separated by newlines (according to the operating system's newline conventions). Blank lines and lines that start with the "#" character (ASCII 35) are considered comments, and are ignored. Long entries may be continued on multiple lines if each non-terminal line ends with a backslash character ('\', ASCII 92), in which case the multiple lines are to be treated as a single mailcap entry. Note that for such "continued" lines, the backslash must be the last character on the line to be continued.

Each mailcap entry consists of a number of fields, separated by semi-colons. The first two fields are required, and must occur in the specified order. The remaining fields are optional, and may appear in any order.

The first field is the content-type, which indicates the type of data this mailcap entry describes how to handle. It is to be matched against the type/subtype specification in the "Content-Type" header field of an Internet mail message. If the subtype is specified as "*", it is intended to match all subtypes of the named content-type.

The second field, view-command, is a specification of how the message or body part can be viewed at the local site. Although the syntax of this field is fully specified, the semantics of program execution are necessarily somewhat operating system dependent.

The optional fields, which may be given in any order, are as follows:

- The "compose" field may be used to specify a program that can be used to compose a new body or body part in the given format. Its intended use is to support mail composing agents that support the composition of multiple types of mail using external composing agents. As with the view- command, the semantics of program execution are operating system dependent. The result of the composing program may be data that is not yet suitable for mail transport – that is, a Content- Transfer-Encoding may need to be applied to the data.
- The "composetyped" field is similar to the "compose" field, but is to be used when the composing program needs to specify the Content-type header field to be applied to the composed data. The "compose" field is simpler, and is preferred for use with existing (non-mail-oriented) programs for composing data in a given format. The "composetyped" field is necessary when the Content-type information must include auxilliary parameters, and the composition program must then know enough about mail formats to produce output that includes the mail type information.
- The "edit" field may be used to specify a program that can be used to edit a body or body part in the given format. In many cases, it may be identical in content to the "compose" field, and shares the operating-system dependent semantics for program execution.
- The "print" field may be used to specify a program that can be used to print a message or body part in the given format. As with the view-command, the semantics of program execution are operating system dependent.
- The "test" field may be used to test some external condition (e.g. the machine architecture, or the window system in use) to determine whether or not the mailcap line applies. It specifies a program to be run to test some condition. The semantics of execution and of the value returned by the test program are operating system dependent. If the test fails, a subsequent mailcap entry should be sought. Multiple test fields are not permitted – since a test can call a program, it can already be arbitrarily complex.
- The "needsterminal" field indicates that the view-command must be run on an interactive terminal. This is needed to inform window-oriented user agents that an interactive terminal is needed. (The decision is not left exclusively to the view-command because in some circumstances it may not be possible for such programs to tell whether or not they are on interactive terminals.) The needsterminal command should be assumed to apply to the compose and edit commands, too, if they exist. Note that this is NOT a test – it is a requirement for the environment in which the program will be executed, and should typically cause the creation of a terminal window when not executed on either a real terminal or a terminal window.
- The "copiousoutput" field indicates that the output from the view-command will be an extended stream of output, and is to be interpreted as advice to the UA (User Agent mail-reading program) that the output should be either paged or made scrollable. Note that it is probably a mistake if needsterminal and copiousoutput are both specified.
- The "description" field simply provides a textual description, optionally quoted, that describes the type of data, to be used optionally by mail readers that wish to describe the data before offering to display it.

- The "x11-bitmap" field names a file, in X11 bitmap (xbm) format, which points to an appropriate icon to be used to visually denote the presence of this kind of data.
- Any other fields beginning with "x-" may be included for local or mailer-specific extensions of this format. Implementations should simply ignore all such unrecognized fields to permit such extensions, some of which might be standardized in a future version of this document.

Advanced Features

Searching

Interfacing to VM

Interfacing to GNUS

Using PGP/PEM

More Help

If you need more help on W3, please send me mail (wmperry@indiana.edu). Several discussion lists have also been created for W3. To subscribe, send mail to <listname>-request@indiana.edu. All other mail should go to <listname>@indiana.edu

- w3-announce – this list is for anyone interested in W3, and should in general only be used by me. The gnu.emacs.sources newsgroup and a few other mailing lists are included on this. You may use this if you have written an enhancement to W3 that you wish more people to know about. (www-announce@info.cern.ch is included on this list).
- w3-beta – this list is for beta testers of W3. These brave souls test out not-quite stable code.
- w3-dev – a list consisting of myself and a few other people who are interested in the internals of W3, and doing active development work. Pretty dead right now, but I hope it will grow.

If you need more help on the World Wide Web in general, please refer to the newsgroup comp.infosystems.www. There are also several discussion lists concerning the Web. Send mail to listserv@info.cern.ch with a subject line of 'subscribe <listname>'. All mail should go to <listname>@info.cern.ch. Administrative mail should go to www-admin@info.cern.ch. The lists are:

- www-talk – for general discussion of the World Wide Web, where its going, new features, etc. All the major developers are subscribed to this list.
- www-announce – for announcements concerning the World Wide Web. Server changes, new servers, new software, etc.

As a last resort, you may always mail me. I'll try to answer as quickly as I can.

Future Directions

Changes are constantly being made to the emacs browser (hopefully all for the better). This is a list of the things that are being worked on right now.

To fix before 2.1

1. Way to save url under point without viewing it first.
2. Fix <DL> handling.

Fix before 2.2

1. more texinfo documentation
2. ability to read in .info files natively
3. cacheing - easy in emacs19/lucid - just grab the entire buffer and tell it to save the text properties, then insert it into a new buffer and boom, done
4. have a 'timeout' when retrieving images. Use (nth 1 (current-time)) when in lucid.
5. PATHs
6. TABLEs
7. Merge hotlist stuff with bookmark.el

Long range goals

1. Multi-DTD browsing
2. Style sheets
3. New parser, with process-filter and process-sentinel

Internals of W3

This chapter will attempt to explain some of the internal workings of W3 and various data structures that are used. It will also detail some functions that are useful for using some of the W3 functionality from within your own programs, or extending the current capabilities of W3.

Generalized ZONES

Due to the many different *flavors* of Emacs in existence, the addition of data and font information to arbitrary regions of text has been generalized. The following functions are defined for using/manipulating these *zones* of data.

`w3-add-zone (start end style data &optional highlight)`
: This function creates a zone between buffer positions start and end, with font information specified by style, and a data segment of data. If the optional argument highlight is non-`nil`, then the region will highlight when the mouse moves over it.

`w3-zone-at (point)`
: Returns the the zone at *point*. Preference is given to hypertext links, then to form entry areas, then to inlined images. So if an inlined image was part of a hypertext link, this would always return the hypertext link.

`w3-zone-data (zone)`
: Returns the zone's data segment. The data structures used in W3 are relatively simple. They are just list structures that follow a certain format. The two main data types are *form objects*, *link objects*,and *inlined images*. All the information for these types of links are stored as lists.

`w3-zone-hidden-p (zone)`
: Returns `t` if and only if a zone is currently invisible.

`w3-hide-zone (start end)`
: Makes a region of text from `start` to `end` invisible.

`w3-unhide-zone (start end)`
: Makes a region of text from `start` to `end` visible again.

`w3-zone-start (zone)`
: Returns an integer that is the start of zone, as a buffer position. In emacs 18.xx, this will return a marker instead of an integer, but it can be used just like an integer.

`w3-zone-end (zone)`
: Returns an integer that is the end of zone, as a buffer position. In emacs 18.xx, this will return a marker instead of an integer, but it can be used just like an integer.

`w3-zone-eq (zone1 zone2)`
: Returns `t` if and only if zone1 and zone2 represent the same region of text in the same buffer, with the same properties and data.

`w3-delete-zone (zone)`
: Removes zone from its buffer (or current buffer). The return value is irrelevant, and varies for each version of emacs.

`w3-all-zones ()`
: Returns a list of all the zones contained in the current buffer. Useful for extracting information about hypertext links or form entry areas. Programs should not rely on this list being sorted, as the order varies with each version of emacs.

`w3-zone-at (pt)`
: This will return the zone at character position PT in the current buffer that is either a link or a forms entry area. Returns nil if no link at point.

These data structures are what is generally returned by `w3-zone-data`.

Global variables

There are also some variables that may be useful if you are writing a program or function that interacts with W3. All of the `w3-current-*` variables are local to each buffer.

`w3-current-mime-headers`
: An assoc list of all the MIME headers for the current document. Keyed on the lowercase MIME header (ie: '`content-type`' or '`content-encoding`'.

`w3-current-server`
: Server that the current document was retrieved from.

`w3-current-file`
: Filename of the current document

`w3-current-type`
: A string representing what network protocol was used to retrieve the current buffer's document. Can be one of http, gopher, file, ftp, news, or mailto.

`w3-current-port`
: Port # of the current document.

`w3-current-last-buffer`
: The last buffer seen before this one.

`w3-running-FSF19`
: This is `t` if and only if we are running in FSF Emacs 19.

`w3-running-epoch`
: This is `t` if and only if we are running in Epoch 4.x

`w3-running-lemacs`
: This is `t` if and only if we are running in Lucid Emacs 19.6 or Lucid Emacs 19.8.

`w3-running-new-lucid`
: This is `t` if and only if we are running in Lucid Emacs 19.9 or later.

Data Structures

Form objects are used to store information about a FORM data entry area.

1. `'w3form`
2. A cons pair of (METHOD . URL), where METHOD specifies what method to use to retrieve the form when it is submitted (ie: '`GET`') and URL is a fully specified URL pointing at where to submit the FORM data to.
3. The type of input area this is. (ie: '`CHECKBOX`' or '`RADIO`')
4. The name of the input tag. This is used when sending the form to the server, so that the server can tell what data is what.
5. The default value of the input area. Gotten from the INPUT tag at creation time.
6. The current value of the input area.
7. Whether the item is checked or not. Only used for RADIO or CHECKBOX items.
8. The size (in characters) of the input area. Not used for CHECKBOX, RADIO, or TEXTAREA input areas.
9. The maximum length of the input. Only used for TEXT or PASSWORD input areas.
10. The form that this input area belongs to. Each form in the same buffer has a unique identifier assigned when the document is parsed. It is used when the form is submitted to get only the data for the correct form.
11. A list of strings that represent the choices for this input area. Only used for SELECT tags.

A new development in the World Wide Web is the concept of collapsible areas of text. If a zone controls one of these regions, it will be marked with the **w3expandlist** property. The format of this structure is:

1. `'w3expandlist`
2. A marker representing the start of the hidden text as a buffer position.
3. A marker representing the end of the hidden text as a buffer position.

A zone with the **w3graphic** property is a link to an inlined image's source file.

1. `'w3graphic`
2. The full URL of the inlined image. This is only ever returned if the inlined image is the only extent under point, or `w3-follow-inlined-image` is invoked.

A zone with the **w3** property is a full-fledged hypertext link to another document.

1. `'w3`
2. The ID attribute of this link. Used for resolving references to specific points within a document (ie: '`file.html#sectionA`'.
3. The HREF attribute of this link. This is a fully specified URL pointing at a network resource. All relative directory references should have been removed before being stored in this structure.
4. The text between the `<A>` and `</A>` tags. This is used to build menus or to get the text of a link without doing a buffer-substring.
5. The URN attribute of this link. Currently not used for anything, waiting for the URN specification to be hammered out.
6. The REL attribute of this link. Specifies the links relevance to the current document.
7. The REV attribute of this link. Specifies the current documents relevance to the link.
8. The METHODS attribute, which tells what methods can be used on this link. (ie: '`GET, HEAD, PUT`'.

Miscellaneous Functions

I have done quite a bit of work trying to make a semi-clean interface to the internals of W3. Here is a list of functions that you can use to take advantage of the World Wide Web.

`w3-clear-tmp-buffer`
: This will set the current buffer to be `w3-working-buffer`, creating it if necessary, and erase it. This should usually be called before retrieving URLs.

`w3-convert-html-to-latex`
: This will take a buffer of HTML markup (which should be in `w3-working-buffer`), and convert it into LaTeX. This is an adaptation of the simple sed scripts from Cern. Does as good a job as the html2latex program, and I usually prefer its formatting over html2latex's.

`w3-extension-to-mime`
: This functions takes a file extension and returns the corresponding MIME-type, as defined in `w3-mime-extensions`. For example, '`(w3-extension-to-mime ".gif")`' would return '`"image/gif"`'.

`w3-fetch`
: This function takes a URL as its only argument. It then attempts to retrieve the URL using the specified method, using data (if any) as the body of the MIME request. For example: '`(w3-fetch "http://cs.indiana.edu/")`' would retrieve the Indiana University CS home page and parse it as HTML.

`w3-fix-entities-in-string`
: This function takes a string, and removes all HTML[+] entity references from it, replacing them with the correct character(s). It consults the variable `w3-html-entities` for the entity names and translations. For example, '`(w3-fix-entities-in-string ">testing<&")`' would return '`">testing<&"`'.

`w3-generate-new-buffer-name`
: This function takes a string, and returns the first unique buffer name using that string as a base. For example '`(w3-generate-new-buffer-name "new-buff")`' would return '`"new-buff<1>"`' if buffer `new-buff` already existed.

`w3-generate-unique-filename`
: This functions returns a string that represents a unique filename in the /tmp directory. For example, '`(w3-generate-unique-filename)`' would return '`"/tmp/w3-tmp129440"`'. The filename is arrived at by using a unique prefix (w3-tmp), the uid of the current user (12944 in my case), and a number that is incremented if a file already exists.

`w3-buffer-visiting (url)`
: Return the name of a buffer (if any) that is visiting URL.

`w3-create-mime-request (fname ref-url)`
: Create a MIME request for the file fname. The Referer: field of the HTTP/1.0 request is set to the value of ref-url if necessary. Returns a string that can be sent to an HTTP server. The request uses several variables that control how the request looks.

  If the value of `w3-current-server` is found in the assoc list of `w3-bad-server-list`, then the basic HTTP/0.9 request type is used. This loses lots of information and server-side typing of files, but it is necessary for some older HTTP/0.9 servers that can't understand the newer, longer HTTP/1.0 requests.

If the value of w3-request-extra-headers is non-nil, then it will be used as extra MIME headers in the

w3-get-url-at-point

This function returns the url at a point specified by an optional argument. If no argument is given to the function (point) is used. Tries to find the url closest to that point, but does not change the users position in the buffer. Has a preference for looking backward when not directly on a URL.

w3-hexify-string

This function takes a string and replaces any characters that are not acceptable in a URL with the "escaped" encoding that is standard for URLs (replaces the character with a % followed by the hexadecimal representation of the ASCII value of the character). For example, '(w3-hexify-string "this is a test")' would return '"this%20is%20a%20test"'.

w3-insert-entities-in-string

This function takes a string and replaces any occurences of HTML[+] reserved characters with the corresponding entity definitions. For example, '(w3-insert-entities-in-string "<testing>")' would return '"<testing>"'.

w3-mime-viewer

This functions takes a string specifying a MIME content-type, and returns a string or symbol. If it returns a string, it specifies a shell command to execute to view that type of file. This string can be passed to 'format with a file name to get the complete command to use. If it is a symbol, it represents an Emacs lisp function to call with no arguments.

w3-open-stream

This function takes the same parameters as open-network-stream, and functions similarly. It takes a process name, a buffer name, a host name, and a port number or server name. It attempts to open a network connection to the remote host on the specified port/service name, with output going to the buffer. It returns the process object that is the network connection. This function is different from open-network-stream in that it allows you to use programs in a subprocess to emulate a network connection. This is necessary if you are behind a firewall, or something similar. If the variable w3-use-telnet is t, then the program specified by w3-telnet-prog will be run in an asynchronous subprocess. This program should expect a command line of the form 'w3-telnet-prog hostname portnumber', and feed anything on its standard input to the remote machine.

w3-retrieve

This function takes 3 arguments, a URL, a method type, and a data block. It then attempts to retrieve the URL using the specified method, using data (if any) as the body of the MIME request. For example: ‘(w3-fetch "http://cs.indiana.edu/" "GET" nil)’ would retrieve the Indiana University CS home page. This function will do **no** parsing of the retrieved page, and leaves you in the buffer containing the document you requested. Any HTTP/1.0 redirection/authorization is done before this function exits.

w3-unhex-string

This is the opposite of w3-hexify-string. It removes any %XXX encoded characters in a string. For example ‘(w3-unhex-string "this%20is%20a%20test")’ would return ‘"this is a test"’.

w3-upcase-region

This function takes a start and end position in the current buffer as its arguments, and converts all the text to uppercase, except for text between < and >, and it also ignores HTML[+] entity references (ie: ‘<’). This is useful for converting regions to uppercase without corrupting any URLs within it.

w3-view-this-url

This function returns the URL of the zone under point (if no zone is under point, then it returns nil). If the optional argument is nil, then the URL is also displayed in the minibuffer.

w3-view-url

This function returns the URL of current document. If the optional argument is nil, then the URL is also displayed in the minibuffer.

Concept Index

Key Index

V

Command Index

Variable Index

Short Contents

Table of Contents