NAME

WWW − World Wide Web Package

SYNOPSIS

```
extract_description( FILE )
extract_meta( FILE, NAME )
hyperlink( LIST )
trim_whitespace( LIST )
url_decode( LIST )
url_encode( LIST )
```

DESCRIPTION

This package is a Perl interface to the World Wide Web, specifically, web servers, to parse HTML form data, extract descriptions of or meta information from files, and hyperlink text.

HTML FORMS

Automatic Processing

Prior to the display of a page, all form data, if any, is processed automatically by this package and the results are deposited in the

```
%FORM
```

Perl hash (in the package `main`). Both the `GET` method as well as both the *application/x-www-form-urlencoded* and *multipart/form-data* `PUT` methods are supported. For example, given the simple search form:

```
<FORM ACTION="search.cgi">
<B>Search:</B>
<INPUT TYPE=text NAME=search SIZE=30>
<INPUT TYPE=submit VALUE="Search">
</FORM>
```

the CGI script can refer to the value the user entered for the search field as `$FORM{ search }`.

File Upload

File uploads, as part of forms with `ENCTYPE` set to *multipart/form-data* are supported. For example, given the form:

```
<FORM METHOD=post ENCTYPE="multipart/form-data" ACTION="upload.cgi">
<INPUT TYPE=file NAME="file_name" SIZE=35>
<INPUT TYPE=submit VALUE="Upload">
</FORM>
```

if the user selects the file `/home/pjl/resume.txt`, only the base name of the file, `resume.txt`, is placed into `$FORM{ file_name }` and the file itself is uploaded to `/tmp/resume.txt` where the CGI script can do with it as it pleases.

SUBROUTINES

The following Perl subroutines are defined and available:

`extract_description(` *FILE* `)`

Extracts a description from an HTML or plain text file given by the *FILE* name; *FILE* should be an absolute path. The first `$description::chars` (default: 2048) characters are read. If the file ends in one of the extensions `htm`, `html`, or `shtml`, it is presumed to be an HTML file; if the file ends in `txt`, it is presumed to be a plain text file. Other extensions are not recognized and no description is returned for them.

For HTML files, first, if a `<META NAME="description" CONTENT="...">` or a `<META`

`NAME="DC.description" CONTENT="...">` (Dublin Core) element is found, then the words specified as the value of the `CONTENT` attribute is returned as the description.

Otherwise, all HTML comments, text between `<SCRIPT>`, `<STYLE>`, and `<TITLE>` tags, and all other HTML tags are stripped. If `<AREA ... ALT="...">` or `<IMG ... ALT="...">` elements are found, then the words specified as the value of the `ALT` attributes are extracted.

Finally, for either HTML or plain text files, at most `$description::words` (default: 50) are returned.

`extract_meta(` *FILE*`,` *NAME* `)`
Extracts the value of the `CONTENT` attribute from a `META` element having the given `NAME` attribute from an HTML file given by the *FILE* name; *FILE* should be an absolute path. The file must end in one of the extensions `htm`, `html`, or `shtml` to be considered an HTML file. The first `$description::chars` (default: 2048) characters are read. The characters are cached between consecutive calls using the same filename.

`hyperlink(` *LIST* `)`
Adds hyperlinks to strings: that is strings that contain substrings that are valid URLs (according to RFC 1630) have the appropriate HTML tags ''wrapped'' around them so that they will be selectable when displayed in a browser. The `ftp`, `gopher`, `http`, `https`, `mailto`, `news`, `telnet`, and `wais` URLs are recognized. Example:

```
Read all about it at
http://www.usatoday.com/
```

becomes:

```
Read all about it at
<A HREF="http://www.usatoday.com/">http://www.usatoday.com/</A>
```

`trim_whitespace(` *LIST* `)`
Trims (removes) all whitespace (including newlines) from both the beginning and end of strings. Whitespace in the middle of strings (including multi-line strings) is not removed.

`url_decode(` *LIST* `)`
Decodes *url-encoded* strings replacing '+' with ' ' and the sequence `%xx` with a single charcter having the ASCII code denoted by *xx* in hexadecimal. This subroutine is called automatically for all processed form data.

`url_encode(` *LIST* `)`
Encodes all ''unsafe'' characters in strings (according to RFC 1738). This subroutine is useful for encoding strings for use in the construction of URLs. Example:

`Is a + b = c?` *becomes* Is+a+%2B+b+%3D+c%3F

SEE ALSO

perl(1)

Tim Berners-Lee. ''Universal Resource Identifiers in WWW,'' *Request for Comments 1630*, Network Working Group of the Internet Engineering Task Force, June 1994.
http://info.internet.isi.edu/in-notes/rfc/files/rfc1630.txt

Tim Berners-Lee, Larry Masinter, and Mark McCahill. *Uniform Resource Locators (URL)*, Network Working Group, Request for Comments 1738, 1994.
http://ds.internic.net/rfc/rfc1738.txt

Larry Masinter and Ernesto Nebel. ‘‘Form-based File Upload in HTML,’’ *Request for Comments 1867,* Network Working Group of the Internet Engineering Task Force, November 1995.
http://info.internet.isi.edu/in-notes/rfc/files/rfc1867.txt

Dave Raggett, Arnaud Le Hors, and Ian Jacobs. ‘‘Notes on helping search engines index your Web site,’’ *HTML 4.0 Specification, Appendix B: Performance,* World Wide Web Consortium, April 1998.
http://www.w3.org/TR/REC-html40/appendix/notes.html#recs

−−. ‘‘Objects, Images, and Applets: How to specify alternate text,’’ *HTML 4.0 Specification, section 13.8,* World Wide Web Consortium, April 1998.
http://www.w3.org/TR/REC-html40/struct/objects.html#h-13.8

Dublin Core Directorate. ‘‘The Dublin Core: A Simple Content Description Model for Electronic Resources.’’
http://purl.oclc.org/dc/

Larry Wall, Tom Christiansen, and Randal L. Schwartz. *Programming Perl, 2nd ed.*, O’Reilly & Associates, Inc., Sebastopol, CA, 1996.
http://www.oreilly.com/

AUTHOR

Paul J. Lucas *<pjl@best.com>*