NAME

URI::Escape − Percent−encode and percent−decode unsafe characters

SYNOPSIS

```
use URI::Escape;
$safe = uri_escape("10% is enough\n");
$verysafe = uri_escape("foo", "\0-\377");
$str  = uri_unescape($safe);
```

DESCRIPTION

This module provides functions to percent-encode and percent-decode URI strings as defined by RFC 3986. Percent-encoding URI's is informally called "URI escaping". This is the terminology used by this module, which predates the formalization of the terms by the RFC by several years.

A URI consists of a restricted set of characters. The restricted set of characters consists of digits, letters, and a few graphic symbols chosen from those common to most of the character encodings and input facilities available to Internet users. They are made up of the "unreserved" and "reserved" character sets as defined in RFC 3986.

```
unreserved    = ALPHA / DIGIT / "-" / "." / "_" / "~"
reserved      = ":" / "/" / "?" / "#" / "[" / "]" / "@"
                "!" / "$" / "&" / "'" / "(" / ")"
              / "*" / "+" / "," / ";" / "="
```

In addition, any byte (octet) can be represented in a URI by an escape sequence: a triplet consisting of the character "%" followed by two hexadecimal digits. A byte can also be represented directly by a character, using the US-ASCII character for that octet.

Some of the characters are *reserved* for use as delimiters or as part of certain URI components. These must be escaped if they are to be treated as ordinary data. Read RFC 3986 for further details.

The functions provided (and exported by default) from this module are:

uri_escape(`$string`)
uri_escape(`$string`, `$unsafe`)

Replaces each unsafe character in the `$string` with the corresponding escape sequence and returns the result. The `$string` argument should be a string of bytes. The **uri_escape()** function will croak if given a characters with code above 255. Use **uri_escape_utf8()** if you know you have such chars or/and want chars in the 128 .. 255 range treated as UTF−8.

The **uri_escape()** function takes an optional second argument that overrides the set of characters that are to be escaped. The set is specified as a string that can be used in a regular expression character class (between [ ]). E.g.:

```
"\x00-\x1f\x7f-\xff"          # all control and hi-bit characters
"a-z"                         # all lower case characters
"^A-Za-z"                     # everything not a letter
```

The default set of characters to be escaped is all those which are *not* part of the `unreserved` character class shown above as well as the reserved characters. I.e. the default is:

```
"^A-Za-z0-9\-\._~"
```

uri_escape_utf8(`$string`)
uri_escape_utf8(`$string`, `$unsafe`)

Works like **uri_escape()**, but will encode chars as UTF−8 before escaping them. This makes this function able to deal with characters with code above 255 in `$string`. Note that chars in the 128 .. 255 range will be escaped differently by this function compared to what **uri_escape()** would. For chars in the 0 .. 127 range there is no difference.

Equivalent to:

```
    utf8::encode($string);
    my $uri = uri_escape($string);
```

Note: JavaScript has a function called **escape()** that produces the sequence "%uXXXX" for chars in the 256 .. 65535 range. This function has really nothing to do with URI escaping but some folks got confused since it "does the right thing" in the 0 .. 255 range. Because of this you sometimes see "URIs" with these kind of escapes. The JavaScript **encodeURIComponent()** function is similar to **uri_escape_utf8()**.

uri_unescape($string,...)

Returns a string with each `%XX` sequence replaced with the actual byte (octet).

This does the same as:

```
   $string =~ s/%([0-9A-Fa-f]{2})/chr(hex($1))/eg;
```

but does not modify the string in-place as this RE would. Using the **uri_unescape()** function instead of the RE might make the code look cleaner and is a few characters less to type.

In a simple benchmark test I did, calling the function (instead of the inline RE above) if a few chars were unescaped was something like 40% slower, and something like 700% slower if none were. If you are going to unescape a lot of times it might be a good idea to inline the RE.

If the **uri_unescape()** function is passed multiple strings, then each one is returned unescaped.

The module can also export the `%escapes` hash, which contains the mapping from all 256 bytes to the corresponding escape codes. Lookup in this hash is faster than evaluating `sprintf("%%%02X", ord($byte))` each time.

SEE ALSO

URI

COPYRIGHT