

# **1 Apache::Util - Interface to Apache C util functions**

## 1.1 Synopsis

```
use Apache::Util qw(:all);
```

## 1.2 Description

This module provides a Perl interface to some of the C utility functions available in Apache. The same functionality is available in libwww-perl, but the C versions are faster:

```
use Benchmark;
timethese(1000, {
    C => sub { my $esc = Apache::Util::escape_html($html) },
    Perl => sub { my $esc = HTML::Entities::encode($html) },
});
```

```
Benchmark: timing 1000 iterations of C, Perl...
    C:  0 secs ( 0.17 usr  0.00 sys =  0.17 cpu)
    Perl: 15 secs (15.06 usr  0.04 sys = 15.10 cpu)
```

```
use Benchmark;
timethese(10000, {
    C => sub { my $esc = Apache::Util::escape_uri($uri) },
    Perl => sub { my $esc = URI::Escape::uri_escape($uri) },
});
```

```
Benchmark: timing 10000 iterations of C, Perl...
    C:  0 secs ( 0.55 usr  0.01 sys =  0.56 cpu)
    Perl:  2 secs ( 1.78 usr  0.01 sys =  1.79 cpu)
```

## 1.3 Functions

- **escape\_html**

This routine replaces unsafe characters in \$string with their entity representation.

```
my $esc = Apache::Util::escape_html($html);
```

This function will correctly escape US-ASCII output. If you are using a different character set such as UTF8, or need more control on the escaping process, use HTML::Entities.

- **escape\_uri**

This function replaces all unsafe characters in the \$string with their escape sequence and returns the result.

```
my $esc = Apache::Util::escape_uri($uri);
```

- **unescape\_uri**

This function decodes all %XX hex escape sequences in the given URI.

```
my $unesaped = Apache::Util::unescape_uri($safe_uri);
```

- **unescape\_uri\_info**

This function is similar to unescape\_uri() but is specialized to remove escape sequences from the query string portion of the URI. The main difference is that it translates the “+” character into spaces as well as recognizing and translating the hex escapes.

Example:

```
$string = $r->uri->query;
my %data = map { Apache::Util::unescape_uri_info($_) }
              split /[=&]/, $string, -1;
```

This would correctly translate the query string  
 ‘name=Fred+Flintstone&town=Bedrock’ into the hash:

```
data => 'Fred Flintstone',
town => 'Bedrock'
```

- **parsedate**

Parses an HTTP date in one of three standard forms:

```
Sun, 06 Nov 1994 08:49:37 GMT ; RFC 822, updated by RFC 1123

Sunday, 06-Nov-94 08:49:37 GMT ; RFC 850, obsoleted by RFC 1036

Sun Nov 6 08:49:37 1994 ; ANSI C's asctime() format
```

Example:

```
my $secs = Apache::Util::parsedate($date_str);
```

- **ht\_time**

Format a time string.

Examples:

```
my $str = Apache::Util::ht_time(time);

my $str = Apache::Util::ht_time(time, "%d %b %Y %T %Z");

my $str = Apache::Util::ht_time(time, "%d %b %Y %T %Z", 0);
```

- **size\_string**

Converts the given file size into a formatted string. The size given in the string will be in units of bytes, kilobytes, or megabytes, depending on the size.

## 1.4 Author

```
my $size = Apache::Util::size_string -s $r->finfo;
```

- **validate\_password**

Validate a plaintext password against a smashed one. Use either `crypt()` (if available), `ap_MD5Encode()` or `ap_SHA1Encode` depending upon the format of the smashed input password.

Returns true if they match, false otherwise.

```
if (Apache::Util::validate_password("slipknot", "aXYx4GnaCrDQc")) {  
    print "password match\n";  
}  
else {  
    print "password mismatch\n";  
}
```

## 1.4 Author

Doug MacEachern

## 1.5 See Also

perl.

## Table of Contents:

1	Apache::Util - Interface to Apache C util functions . . . . .	1
1.1	Synopsis . . . . .	2
1.2	Description . . . . .	2
1.3	Functions . . . . .	2
1.4	Author . . . . .	4
1.5	See Also . . . . .	4